Open Geospatial Consortium Inc.

Date: 2007-07-17

Reference number of this document: **OGC® 07-000**

Version: 1.0.0

Category: OpenGIS® Implementation Specification

Editor: Mike Botts
Co-Editor: Alexandre Robin
University of Alabama in Huntsville

OpenGIS® Sensor Model Language (SensorML) Implementation Specification

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OpenGIS® Publicly Available Standard
Document subtype: Implementation Specification
Document stage: Final
Document language: English

(Page intentionally left blank)

Table of Contents

i. Preface

The primary focus of SensorML is to define processes and processing components associated with the measurement and post-measurement transformation of observations.

This specification is one of five engineering specifications produced under OGC's Sensor Web Enablement (SWE) activity, which is being executed under OGC's Interoperability Program. The initial version was produced during OGC Web Services (OWS) 1.1 Initiative, conducted in 2001. The previous version was produced under the OGC Web Services (OWS) 3.0 Initiative, conducted March 2005 - October 2005. This version is in response to recommendations during the Release for Public Comment (April – August 2006), as well as efforts that have been continued since the OWS 3 initiatives and during the OWS 4 Initiative, conducted May 2006 – December 2006. This document provides version 1.0 of the SensorML core specification and supersedes all previous document versions including 05-086r3.

ii. Document Terms and Definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 05-008], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word "shall" (not "must") is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

iii. Submitting Organizations

The following organizations submitted this document to the Open Geospatial Consortium, Inc:

a. University of Alabama in Huntsville
b. Commonwealth Scientific and Industrial Research Organisation (CSIRO) Australia
c. IRIS Corporation
d. Distributed Instruments
e. Galdos Systems, Inc
f. Image Matters, LLC
g. SeiCorp, Inc.

iv. Submission Contact Point

All questions regarding this document should be directed to the editor or the contributors:

CONTACT	COMPANY	ADDRESS
Mike Botts (Editor)	University of Alabama in Huntsville	ESSC / NSSTC Huntsville, AL 35899
Alexandre Robin	University of Alabama in Huntsville	ESSC / NSSTC Huntsville, AL 35899
Simon Cox	CSIRO, Australia	PO Box 1130, Bentley WA 6102 Australia
Steve Havens	IRIS Corporation	4220 Varsity Drive Suite E Ann Arbor, MI 48108
Jeff Ricker	Distributed Instruments	33 West Shore Drive Putnam Valley, NY 10579
Ron Lake	Galdos Systems, Inc.	Suite 200, 115 West Pender Street, Vancouver, B.C. V6E 2P4
Harry Niedzwiadek	Image Matters, LLC	214 South King St. Leesburg VA 20175
Bill Craig	SeiCorp, Inc.	5870 Trinity Parkway, Suite 350 Centreville, VA 20120-1970

v. Revision History

Date	Release	Author	Section modified	Description
2006-06-01	05-086r3	meb		Version 0.0, approved in July 2006
2007-01-10	07-000	bc	throughout	Extensive editorial corrections
2007-01-10	07-000	meb, ar	throughout	Reflects several changes to the schema based on recommendations during the RFC process: (1) derivation from GML 3.1.1, (2) modifications to SWE Common data types to better reflect ISO data models, (3) better separation of physical and non-physical processes
2007-03-02	07-000	sc, meb	8.4, 8.6, 10.2.6, 10.2.7	Added models and examples for Time Aggregates and Phenomenon
2007-03-02	07-000	meb	8.7, 10.2.3	Added ObservableProperty as by-reference only simple type for measureably properties of Phenomenon or ontologies
2007-03-02	07-000	meb	throughout	Reorganized sections, splitting SWE Common sections from SensorML sections
2007-03-06	07-000	meb, body, reynolds	throughout	General editing
2007-03-30	07-000	meb, sc	8.0	Revised UML models for SWE Common
2007-05-04	07-000	meb	throughout	Replaced “role” attribute with “arcrole”
2007-05-04	07-000	meb	9.2.5	corrected the curve example
2007-07-17	07-000	meb	throughout	corrected URNs in examples to reflect current design

Any issues in this specification are captured in the following format:

Issue Name: [Issue Name goes here. (Your Initials, Date)]
Issue Description: [Issue Description.]
Resolution: [Insert Resolution Details and History.] (Your Initials, Date)]

vi. Recommended Changes to the OpenGIS Abstract Specification

It is recommended that OGC Abstract Specifications dealing with spatial referencing (e.g. OGC Topic 1 – ISO 19107) be revisited to provide more robust definitions of location, position, orientation, coordinate system, and coordinate reference system. The current definitions are heavily "point-biased" rather than "body-bias". Precise georeferencing of sensor observations requires the potential disclosure of the full state of a body, including location and orientation, as well as possibly velocity, acceleration, angular velocity, and angular acceleration. The current definitions of coordinate systems in these specifications ignore the role of the Coordinate System (CS) axes in supporting rotations. Furthermore, the definition of coordinate reference system depends on a "datum" yet the definition of a datum is not provided.

In addition, the robust georeferencing of sensor observations depends on the ability to robustly specify the dynamic state of one body to another (e.g. the state of a sensor to its platform, or to other sensors). The concepts for Coordinate Reference Systems (CRS) within the Abstract Specifications have been derived first to support geodetic coordinates and datums, with engineering or local CRSes being defined relative to these. It is recommended that CRS definitions and concepts begin with more general mathematical principals for axes and coordinate systems, and that the definitions for geodetic CRSes be defined as a specialization of general principals. Furthermore, it is recognized that a robust specification of one body to another (or to a geodetic CRS) requires that one first define a local CRS for the local object and that the state of the local object is thus defined by relating the local CRS to the external CRS. These principals are not considered in the current Abstract Specifications.

vii. Future Work

In future versions of this work, it is planned to:

1. make the UML conformant with ISO 19103, ISO 19118 and ISO 19136 Annex E
2. harmonize between the SWE Common datatypes model and ISO 19103 and ISO 19123
3. harmonize between SensorML and the sensor model to be defined in ISO 19130

4. refactor the “SWE Common” components into a separate document
5. invesitigate harmonization between SensorML and OGC Web Processing Service (WPS)
6. harmonize between SensorML and TransducerML
7. further harmonization of metadata with ISO 19115 and 19139
8. harmonize between SensorML event/history with ISO 19108 and 19109
9. harmonize between SensorML and the Abstract Specification documents for spatial positioning

Foreword

This specification originated under NASA Advanced Information Systems Technology (AIST) funding and has been further advanced under the OGC OWS 1.1, OWS 1.2, OWS 3, and OWS 4 initiatives, and through separate funding from the National Geospatial-Intelligence Agency, Northrop Grumman Corporation TASC, Defense Intelligence Agency, NASA, European Space Agency, and SAIC.

Additional information is available at http://vast.uah.edu/SensorML.

In addition, one can subscribe to the SensorML forum and listserver at:

http://mail.opengeospatial.org/mailman/listinfo/sensorML.

This document includes four annexes; Annexes A and B are normative, while Annexes C and D are informative.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The OGC shall not be held responsible for identifying any or all such patent rights.

Introduction

The Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) activity, is being executed through the OGC Web Services (OWS) initiatives (under the Interoperability Program). The SWE initiatives are establishing the interfaces and protocols that will enable a "Sensor Web" through which applications and services will be able to access sensors of all types over the Web. These initiatives have defined, prototyped and tested several foundational components needed for Sensor Web Enablement, namely:

1. **Sensor Model Language (SensorML)** – The general models and XML encodings for sensors and observation processing. SensorML originated under OWS 1.1, was significantly enhanced under OWS 1.2, OWS 3, and OWS 4 Initiatives.
2. **Observations & Measurements (O&M)** - The general models and XML encodings for sensor observations and measurements. O&M originated under OWS 1.1 and was significantly enhanced under OWS 1.2, OWS 3, and OWS 4.
3. **TransducerML (TML)** – A model and encoding for streaming multiplexed data from a sensor system, and for describing the system and data encoding. Developed initially by the IRIS Corporation, this service is being brought into the OGC SWE framework and has undergone further development as part of the OWS 3 and OWS 4 Initiatives.
4. **Sensor Observation Service (SOS)** – A service by which a client can obtain observations from one or more sensors/platforms (can be of mixed sensor/platform types). Clients can also obtain information that describes the associated sensors and processes. This service originated under OWS 1.1 and was significantly enhanced under OWS 1.2, OWS 2, OWS 3, and OWS4.
5. **Sensor Planning Service (SPS)** – A service by which a client can determine collection feasibility for a desired set of collection requests for one or more mobile sensors/platforms, or a client may submit collection requests directly to these sensors/platforms. This service was defined under OWS 1.2 and enhanced under OWS 3 and OWS 4.
6. **Sensor Alert Service (SAS)** – A service for advertising, subscribing to, and publishing alerts to alert listener clients. This service is being defined under the Sensor Alert Service Interoperability Experiment and enhanced under the OWS 4 Initiative.
7. **Web Notification Service (WNS)** – A service by which a client may conduct asynchronous dialogues (message interchanges) with one or more other services. This service is useful when many collaborating services are

required to satisfy a client request, and/or when significant delays are involved is satisfying the request. This service was defined under OWS 1.2 in support of SPS operations. WNS has broad applicability in many such multi-service applications.

This document specifies SensorML, as well as several SWE Common data components utilized throughout the SWE framework. The other SWE encodings and web services components are specified under separate documents.

While SensorML serves as a component within the OGC Sensor Web Enablement framework, SensorML does not depend upon the presence of the other SWE components. It is envisioned that SensorML will be utilized both as part of, and independent of, the OGC Sensor Web Enablement framework.

Sensor Model Language: An Implementation Specification

1 Scope

This document specifies models and XML encoding for the core SensorML, as well as the definition of several SWE Common data components utilized throughout the SWE framework. The primary focus of SensorML is to define processes and processing components associated with the measurement and post-measurement transformation of observations.

The purposes of SensorML are to:

- Provide descriptions of sensors and sensor systems for inventory management
- Provide sensor and process information in support of resource and observation discovery
- Support the processing and analysis of the sensor observations
- Support the geolocation of observed values (measured data)
- Provide performance characteristics (e.g., accuracy, threshold, etc.)
- Provide an explicit description of the process by which an observation was obtained (i.e., it's lineage)
- Provide an executable process chain for deriving new data products on demand (i.e., derivable observation)
- Archive fundamental properties and assumptions regarding sensor systems

SensorML provides a common framework for any process and process chain, but is particularly well-suited for the description of sensor and systems and the processing of sensor observations. Within SensorML, sensors and transducer components (detectors, transmitters, actuators, and filters) are all modeled as processes that can be connected and participate equally within a process chain or system, and which utilize the same process model frame as any other process.

Processes are entities that take one or more inputs and through the application of well-defined methods using specific parameters, results in one or more outputs. The process model defined in SensorML can be used to describe a wide variety of processes, including actuators, spatial transforms, and data processes, to name a few. SensorML also supports linking between processes and thus supports the concept of process chains, which are themselves defined as processes.

SensorML provides a framework within which the geometric, dynamic, and observational characteristics of sensors and sensor systems can be defined. There are a great variety of sensor types, from simple visual thermometers to complex electron microscopes and earth observing satellites. These can all be supported through the definition of atomic process models and process chains.

The models and schema within the core SensorML definition (i.e., within this document) provide a "skeletal" framework for describing processes, process chains, and sensor

systems. In order to achieve interoperability within and between various sensor communities, implementation of SensorML will require the definition of community-specific semantics (within online dictionaries or ontologies) that can be utilized within the framework. In addition, we envision the development of small, general-use, atomic processes that can serve as components within process chains. This will include a general model and XML profile to be used for defining most if not all detectors, actuators, and filters within a sensor system, as well as general process definitions for fundamental spatial transforms and basic signal processing components. These can also include more specific models used within the remote sensing community, for example, including rigorous and polynomial sensor geolocation models.

Within SensorML, all processes and components are encoded as application schema of the Feature model in the Geographic Markup Language (GML) Version 3.1.1.

With regard to Observations and Measurements, we assume four fundamental components of information:

1. There are properties of physical entities and phenomena that are capable of being measured and quantified. Within the SWE context, a property that is capable of being measured is considered as an "ObservableProperty" or "Phenomenon". Each of these can be referenced in an online dictionary or ontology. Such definitions might include, for example, properties such as temperature, count, rock type, chemical concentration, or radiation emissivity. Within SensorML, such definitions will typically be referenced through URIs (particularly URNs).
2. There are sensors that are capable of observing and measuring particular properties. Either by design or as a result of operational conditions, these sensors have particular response characteristics that can be used to determine the values of the measurements, as well as to assess the quality of these measurements. In addition to the response characteristics, these sensor systems have properties of location and orientation that allow one to associate the measured values with a particular geospatial location at a particular time. The role of the *SensorML* is to provide characteristics required for processing, georegistering, and assessing the quality of measurements from sensor systems.

   Within this context, SensorML documents can serve two possible roles. The first is to describe the procedure by which an existing observation was obtained. This would include the sensor measurement process, as well as any post processing of the raw observations. The second possible role is to provide processing chains with which SensorML-enabled software could derive new data from existing observations on-demand. One might consider this as a "Derivable Observation", since the values don't exist prior to execution of the processing chain.
3. Finally, there are data values that are returned by a sensor system or are derived from sensor measurements. These measurements may be accessed directly from the sensor, or from caches or data stores that distribute and possibly process these data into various products. The processing and georegistration of these measured values require knowledge of the properties of the sensor system. Within the context of the OGC Sensor Web Enablement framework, values returned by

sensors can be provided within the Observations and Measurements schemas or other data provider types.

Within the *Observation* model provided by the Observations and Measurements schema, a procedure describes the processes of measurement and post-measurement processing that resulted in the observation. This procedure can be encoded within SensorML and can be referenced using the *procedure* property of an Observation. Furthermore, a sensor can observe one or more features of interest during a given observation event; these features can be defined with the *featureOfInterest* property of an *Observation*. The location of these features, which may perhaps include a swath of the earth's surface observed by a remote sensor, can be provided by a SensorML process.

All three of these components are linked within the Sensor Web Enablement concepts. While these links will be discussed within this document, only the second component is within the scope of this document.

SensorML can, but generally does not, provide a detailed description of the hardware design of a sensor. Rather it is a general schema for describing ***functional*** *models of the sensor.* The schema is designed such that it can be used to support the processing and geolocation of data from virtually any sensor, whether mobile or dynamic, in-situ or remotely sensed, or active or passive. This allows one to develop general, yet robust, software that can process and geolocate data from a wide variety of sensors, ranging from simple to complex sensor systems.

SensorML enables robust definitions of ***sensor models*** *for providing geolocation of observations from remote sensors. SensorML supports both rigorous geolocation models and mathematical geolocation models.* A *rigorous sensor model* is defined here as one that describes the geometry and physical dynamics of the instrument and provides the ability to utilize this information along with position and orientation of the platform in order to derive geolocation of the sensor data. *Mathematical sensor models* are typically derived using a rigorous model, perhaps augmented by human interaction. These general mathematical models typically hide the physical characteristics of the sensor and allow for geolocation of sensor data through the use of polynomial functions. Different mathematical models can be designed to define a sample location within a variety of coordinate systems, including the local sensor frame, the local frame for the associated platform, or a geographic coordinate reference frame.

Within SensorML, one may choose to model a sensor platform as a system, with its own Coordinate Reference System, to which on-board sensor positions can be referenced. One may also choose to provide relative positions between various sensors while ignoring the platform reference frame, by defining any sensor position relative to an onboard GPS sensor and an orientation (gimbal) sensor. *Either way, for the case of rigorous sensor models, we allow one to separate the description of the sensor from that of its platform. Common platforms include: ground stations, automobiles, aircraft, earth-orbiting satellites, ocean buoys, ships, and people. A deployed sensor is mounted on a static or dynamic platform (or an assembly of nested platforms).*

There are several data component definitions that were previously defined jointly within the SensorML and O&M specifications. These common components are utilized

throughout the SWE encodings and web services. For this reason, these have been separated from the SensorML namespace (namespace prefix: "*sml*"), and have been defined within a "SWE Common" namespace (namespace prefix: "*swe*"). These are defined within the current document, although it is anticipated that the definition (but not the use) of the SWE Common components will be removed from future versions of SensorML and will instead be defined in a separate document.

2 Conformance

Conformance and Interoperability Testing for SensorML may be checked using all the relevant tests that pertain to Annex A (normative). The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in ISO 19105: Geographic information — Conformance and Testing.

3 Normative references

The following normative documents contain provisions, which through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax. (August 1998)*

ISO 8601:2004, *Data elements and interchange formats — Information interchange Representation of dates and times*

ISO/IEC 11404:1996, Information technology — Programming languages, their environments and system software interfaces – Language-independent datatypes

ISO 19101:2002, *Geographic Information--ReferenceModel*

ISO/TS 19103:2005, *Geographic Information — Conceptual schema language*

ISO 19107:2003, *Geographic Information — Spatial schema*

ISO 19108:2002, *Geographic Information — Temporal schema*

ISO 19109:2005, *Geographic Information — Rules for application schemas*

ISO 19115:2003, *Geographic Information — Metadata*

ISO 19118:2005, *Geographic Information — Encoding*

ISO 19123:2005, *Geographic Information — Coverages*

ISO FDIS 19136:2006, *Geographic Information — Geography Markup Language*

ISO FDTS 19139:2006, *Geographic Information — Metadata — XML schema Implementation*

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax. (August 1998)*

OpenGIS® Implementation Specification *Filter Encoding version 1.1. OGC Document 04-095* http://www.opengeospatial.org/

OpenGIS® Implementation Specification *Geography Markup Language, version 3.2. OGC Document 06-XXX* http://www.opengeospatial.org/

OpenGIS® Abstract Specification *Observations and Measurements – Part 1: Observation Schema, OGC document 07-022.*

UCUM, Unified Code for Units of Measure, Schadow, G. and McDonald, C. J. (eds.), http://aurora.rg.iupui.edu/UCUM

W3C XLink, *XML Linking Language (XLink) Version 1.0. W3C Recommendation (27 June 2001)*

W3C XML, *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation (6 October 2000)*

W3C XML Namespaces, *Namespaces in XML. W3C Recommendation (14 January 1999)*

W3C XML Schema Part 1, *XML Schema Part 1: Structures. W3C Recommendation (2 May 2001)*

W3C XML Schema Part 2, *XML Schema Part 2: Datatypes. W3C Recommendation (2 May 2001)*

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1 Actuator

A type of transducer that is a simple element that converts a signal to some action or real world phenomenon. In SensorML a actuator is a particular type of Process Model.

4.2 Coordinate Reference System (CRS)

A spatial or temporal framework within which a position and/or time can be defined. According to ISO 19111, a **coordinate system** that is related to the real world by a **datum**.

4.3 Coordinate System (CS)

According to ISO19111, a set of (mathematical) rules for specifying how coordinates are assigned to points. In this document, a **Coordinate System** is extended to be defined as a set of axes with which **location** and **orientation** can be defined.

4.4 Datum

Undefined in ISO 19111. Defined here as a means of relating a **coordinate system** to the real world by specifying the physical location of the coordinate system and the orientation of the axes relative to the physical object. For a geodetic datum, the definition also includes a reference ellipsoid that approximate the surface of the planetary body.

4.5 Detector

Atomic part of a composite **Measurement System** defining sampling and response characteristic of a simple detection device. A detector has only one input and one output, both being scalar quantities. More complex **Sensors**, such as a frame camera, which are composed of multiple detectors can be described as a detector group or array using a **System** or **Sensor**. In SensorML a detector is a particular type of **Process Model**.

4.6 Determinand

A Parameter or a characteristic of a phenomenon subject to **observation.** Synonym for **observable.** *[O&M]*

4.7 Location

A point or extent in space relative to a **coordinate system**. For point-based systems, this is typical expressed as a set of n-dimensional coordinates within the **coordinate system**. For bodies, this is typically expressed by relating the translation of the origin of an object's local **coordinate system** with respect to the origin of an external reference **coordinate system**.

4.8 Location Model

A model that allows one to locate objects in one local **reference frame** relative to another **reference frame**.

4.9 Measurand

Physical parameter or a characteristic of a phenomenon subject to a **measurement,** whose value is described using a Measure (ISO 19103). Subset of **determinand** or **observable**. *[O&M]*

4.10 Measure (noun)

Value described using a numeric amount with a scale or using a scalar reference system *[ISO/TS 19103]*. When used as a noun, measure is a synonym for physical quantity.

4.11 Measurement (noun)
An **observation** whose result is a **measure** *[O&M]*

4.12 Measurement (verb)
An instance of a procedure to estimate the value of a natural phenomenon, typically involving an instrument or sensor. This is implemented as a dynamic feature type, which has a property containing the result of the measurement. The measurement feature also has a location, time, and reference to the method used to determine the value. A measurement feature effectively binds a value to a location and to a method or instrument.

4.13 Observable, Observable Property (noun)
A parameter or a characteristic of a **phenomenon** subject to **observation.** Synonym for **determinand.** *[O&M]*

4.14 Observation (noun)
An act of observing a property or phenomenon, with the goal of producing an estimate of the value of the property. A specialized event whose result is a data value. *[O&M]*

4.15 Observed Value
A **value** describing a natural phenomenon, which may use one of a variety of scales including nominal, ordinal, ratio and interval. The term is used regardless of whether the value is due to an instrumental observation, a subjective assignment or some other method of estimation or assignment. [O&M]

4.16 Orientation
The rotational relationship of an object relative to a coordinate system. Typically expressed by relating the rotation of an object's local coordinate system relative to an external reference coordinate system.

4.17 Phenomenon
A physical property that can be observed and measured, such as temperature, gravity, chemical concentration, orientation, number-of-individuals.

A characteristic of one or more feature types, the value for which must be estimated by application of some procedure in an observation.

4.18 Position
The location and orientation of an object relative to a coordinate system. For body-based systems (in lieu of point-based systems) is typically expressed by relating the object's local coordinate system to an external reference coordinate system. This definition is in contrast to some definitions (e.g. ISO 19107) which equate position to location.

4.19 Process
A process that takes one or more inputs, and based on parameters and methodologies, generates one or more outputs.

4.20 Process Method
Definition of the behavior and interface of a **Process**. It can be stored in a library so that it can be reused by different **Process** instances (by using 'xlink' mechanism). It essentially describes the process interface and algorithm, and can point the user to existing implementations.

4.21 Process Chain
Composite processing block consisting of interconnected sub-processes, which can in turn be **Process Models** or **Process Chains**. A process chain also includes possible data sources as well as connections that explicitly link input and output signals of sub-processes together. It also precisely defines its own inputs, outputs and parameters.

4.22 Reference Frame
A coordinate system by which the position (location and orientation) of an object can be referenced.

4.23 Result
an estimate of the value of some property generated by a known procedure *[O&M]*

4.24 Sample
A subset of the physical entity on which an observation is made.

4.25 Sensor
An entity capable of observing a phenomenon and returning an observed value. In SensorML, modeled as a specific type of **System** representing a complete **Sensor**. This could be for example a complete airborne scanner which includes several **Detectors** (one for each band).

4.26 Sensor Model
In line with traditional definitions of the remote sensing community, a sensor model is a type of Location Model that allows one to georegister observations from a sensor (particularly remote sensors).

4.27 (Sensor) Platform
An entity to which can be attached sensors or other platforms. A platform has an associated local coordinate frame that can be referenced to an external coordinate reference frame and to which the frames of attached sensors and platforms can be referenced.

4.28 System
Composite model of a group or array of components, which can include detectors, actuators, or sub-systems. A **System** relates a **Process Chain** to the real world and therefore provides additional definitions regarding relative positions of its components and communication interfaces.

4.29 Transducer
An entity that receives a signal as input and outputs a modified signal as output. Includes detectors, actuators, and filters.

4.30 value
member of the value-space of a datatype. A value may use one of a variety of scales including nominal, ordinal, ratio and interval, spatial and temporal. Primitive datatypes may be combined to form aggregate datatypes with aggregate values, including vectors, tensors and images *[ISO11404]*.

5 Conventions

5.1 Symbols (and abbreviated terms)

The following symbols and abbreviated terms are used in this document.

AIRDAS	Airborne Infrared Disaster Assessment System
AIST	Advanced Information Systems Technology
API	Application Programming Interface
CCD	Charge-Coupled Device
CEOS	Committee for Earth Observation Satellites
CPU	Central Processing Unit
CRS	Coordinate Reference System
DDMS	Department of Defense Discovery Metadata Specification
DN	Digital Number
ECEF	Earth-Centered Earth-Fixed
ECI	Earth Centered Inertial
EOS	Earth Observing System
GML	Geographic Markup Language
gml:*	namespace prefix for the GML schema
GMTT	Global Mapping Task Team
GPS	Global Positioning System
IC ISM	Intelligence Community Information Security Marking
IEC	International Electrotechnical Commission
IERS	International Earth Rotation and Reference Systems
IMU	Inertial Measurement Unit
I/O	Input/Output
IP	Interoperability Program
ISO	International Organization for Standardization
JPL	Jet Propulsion Laboratory
MASINT	Measurements and Signals Intelligence
MPP	Military Pilot Project
NASA	National Aeronautics and Space Administration

NAIF	Navigation and Ancillary Information Facility
NGA	National Geospatial-Intelligence Agency
OGC	Open Geospatial Consortium
om:*	namespace prefix for the Observations and Measurement schema
OLS	Optical Line Scanner
OSI	Open System Interconnection
OWS	OGC Open Web Services
O&M	Observations and Measurements
RGB	Red/Green/Blue
RPC	Rapid Position Coordinates
SAS	Sensor Alert Service
SensorML	Sensor Model Language
sml:*	namespace prefix for the SensorML schema
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
swe:*	namespace prefix for the SWE Common schema
TML	Transducer Markup Language
UAH	University of Alabama in Huntsville
uom	Unit(s) of measure
UCUM	Unified Code for Units of Measure
UML	Unified Modeling Language
URI	Universal Resource Identifier?
URN	Universal Resource Name?
VAST	VisAnalysis System Technologies
WGS84	World Geodetic System 1984
WNS	Web Notification Service
XML	eXtensible Markup Language
xs:*	namespace prefix for XMLSchema.2002

5.2 UML Notation

The diagrams that appear in this document are presented using the Unified Modeling Language (UML) static structure diagram. The UML notations used in this document are described in the diagram below.

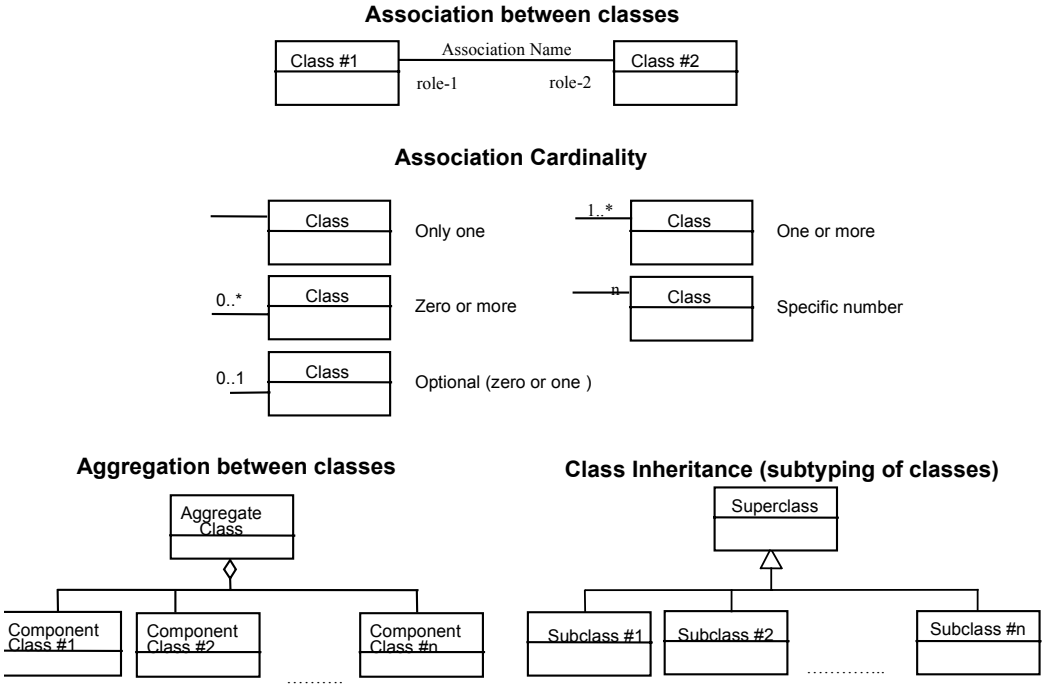


Figure 5.1 — UML notation

The following three stereotypes of UML classes are used:

a) <<Interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes.

b) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.

c) <<Enumeration>> A data type whose instances form a list of alternative literal values. Enumeration means a short list of well-understood potential values within a class.

d) <<CodeList>> A flexible enumeration for expressing a long list of potential alternative values. If the list alternatives are completely known, an enumeration shall be used; if the only likely alternatives are known, a code list shall be used.

e) <<Type>> A stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A Type class may have attributes and associations.

In this document, the following standard data types are used:

a) CharacterString – A sequence of characters

b) Integer – An integer number

c) Double – A double precision floating point number

d) Float – A single precision floating point number

6 Background

6.1 Motivation

While sensors play a fundamental part of our everyday lives, as well as play vital roles within the environmental, intelligence, emergency management, and defense communities, the state of the art is of heterogeneous networks of disparate sensors. These sensors are rarely easily discoverable, and access and processing of observations from these sensors are often confined to stovepipe systems. Even if interoperability has been achieved to some degree within a given sensor community, it usually comes at a cost of inflexible turnkey systems that are expensive and difficult to maintain and extend. The ability to easily discover, access, and process sensors existing within different communities is very rare. SensorML provides a common framework for describing virtually any sensor system, as well as the processing that might be associated with these sensor systems.

The importance of long-term monitoring of the Earth's environment and the development of improved data processing techniques, has raised awareness of the need for preserving low-level sensor data and the information required for reprocessing this data. Unfortunately, such information is often lost or difficult to find five to ten years after completion of a sensor's original mission life. The proposed SensorML is one step toward preserving part of the vital information. required for geolocation and processing of sensor data for both real-time and archival observations.

Often one is unaware of private or public sensor systems that are available for a particular application. Particularly in disaster prevention or remediation, it is vital that one be able to discover and gather observations of relevance from any appropriate sensors in the affected area. SensorML provides a standard means by which sensor and platform capabilities and properties can be published and discovered. As will be discussed in more detail below, SensorML also provides information that allows for geolocation and processing of these sensor observations without a priori knowledge of the sensor's properties.

Web-enabled sensors provide the technology to achieve rapid access to accurate environmental information from the field. Streaming sensor information in standard formats facilitates integration, analysis, and creation of various data "views" that are more meaningful to the end user and avoids the drawbacks of locating and accessing incompatible archived data. This provides a significant advantage in that it reduces the time lag between making measurements and applying those measurements in decision-making. Time savings are particularly noticeable in the management of time critical events such as emergency response, advanced warning systems, and forecasting. A second benefit is in the routine use of data for everyday decision-making. Together, these developments will advance the realization of an integrated, yet distributed, monitoring and assessment system used by government, researchers, businesses, and the public in improving decision making based on high quality, near real time data and information.

Furthermore, recent research and development activities have demonstrated several significant benefits of providing on-demand sensor geolocation within desktop or on-board tools. These include:

- Significant reduction of distributed data from remote sensors; large data volumes resulting from the distribution and storage of per-pixel latitudes and longitudes, as well as other pre-processed geometric relationships can be replaced with the calculation of these values on-demand;
- Improved capabilities for visually integrating and analytically comparing multi-sensor data;
- The ability to more easily correct geolocation errors from within the end-user tools and to redistribute these corrections to the user community;
- The ability to take advantage of several adaptive methods in computer graphics for improving interactivity within visualization tools; and
- Greatly improved capabilities for search and query of spatial-temporal sensor data without the need to request and perhaps store large data sets

SensorML supports the ability to describe the process by which an observation is measured or derived, and can therefore provide the lineage of a sensor observation or simulation results. Using the same framework, SensorML also provides the ability to provide executable process chains for deriving higher-level information. This might support, for example, on-the-fly geolocation of remote sensor observations or on-demand processing of raw data into more meaningful results.

Traditionally, the geolocation and processing of low-level sensor data has required writing or utilizing software specifically designed for that sensor system. The availability of a standard model language for describing platform position and rotation, as well as instrument geometry and dynamics, allows for the development of generic multi-purpose software that can provide geolocation for potentially all remotely sensed data. The availability of such software in turn provides a simple, single Application Programming Interface (API) for tool developers to incorporate sensor geolocation and processing into their application software.

One intent of the SensorML standard is to allow the development of software libraries that can parse these files and calculate required look angles and timing for each sensor pixel. Other efforts are establishing standards for storage and transmission of sensor platform location and rotation in order to insure that such formats are also maintained, available, and readable by similar APIs.

6.2 Importance to archival needs

A standard description format for sensors is important for the long-term definition of the sensor model's fundamental characteristics and assumptions for use in future reprocessing and refining of sensor data.

We are currently entering an era of Earth observation in which we have realized the importance of long-term observation of the Earth's environment. Thus, archiving the raw or low-level sensor data for future reprocessing has taken on greater importance. Equally important is that we preserve the characteristic metadata and assumptions required to reprocess the sensor data. The characteristic data include what is needed for geolocation, calibration, and radiometric processing of the remotely sensed data. Simply archiving the

latitude and longitude values will not only be expensive, but will also prove to be highly inadequate. It is anticipated that further efforts within organizations such as the CEOS Data Subgroup, ISO TC211, and the Open Geospatial Consortium, will be directed toward insuring proper standardization and archiving of other required data, such as platform position and rotation, and target or planet models. This current paper is directed specifically at the adequate description and standardization of fundamental geometric, dynamics, and measurement characteristics of the sensor.

As an example, sensor look angles have traditionally been either pre-calculated and stored within a data array structure, or calculated as needed within software systems developed specifically for that sensor. With time, unfortunately, the actual parameter values for the geometric and dynamic characteristics of the sensor are often lost as contract reports and software become obscure, and as look angle arrays and hardwired software prove difficult to deconvolve into the characteristic sensor parameters. Once the initial mission has been completed and the processing teams dispersed, reprocessing, correction, or refinement of the sensor data thus become very difficult, if not impossible. For example, this was the case for reprocessing of the archived Optical Line Scanner (OLS) data (Ken Knowles/University of Colorado, personal communication), as well as for the 20 year-old data from the Viking Mission to Mars (Bill Taber/JPL, personal communication).

6.3 Importance to software support

The standardization and the availability of SensorML documents for all sensors will allow for significant opportunities for software systems to support the processing, analysis, and visual fusion of multiple sensors.

Traditionally software that supported multiple sensors has been forced to deal with proprietary software designed for each individual sensor. When such software systems still exist and can be located, the software developer is often faced with trying to merge incompatible software architectures and development languages, or with rewriting the software to meet the requirements of his or her software. Even then, the addition of each new sensor system that the developer wishes to support requires the development of individual software modules specific to that sensor, often resulting in redundant code for manipulating and transforming the data.

In contrast, SensorML is based on the concept that the availability of standard sensor system descriptions allows for the development of general processing software capable of geolocating and transforming any sensor data for which such a description exists. This concept is built around the availability of description files for providing sensor-system specific information regarding platform position and rotation, instrument geometry and dynamics, target planet shape and position, and perhaps other time-tagged information, such as data dropouts, instrument modes of operation, or spacecraft clock adjustments.

This concept is dependent on the availability of generic software for parsing these files, and calculating the transformations required to geolocate, and execute processing of the sensor data. A significant advantage of this concept for the developer and ultimately the end user, is that it provides a single source, single application programming interface, for

the geolocation and processing of any sensor system, rather than requiring the developer to locate and implement proprietary software for each sensor system.

In addition, this concept allows for the development of tools that can provide on-demand geolocation and mapping. As SensorML-enabled application software becomes more common, there will be less need to store and distribute volumetrically costly latitude, longitude, altitude, and incident angles values per pixel. Furthermore, correction of sensor geolocation requires only the redistribution of much smaller description files, rather than the redistribution of large collections of reprocessed sensor data. In fact, correction or refinement of geolocation can be conducted by the end user as necessary, rather than relying strictly on the instrument team. Finally, the ability to provide spatial-temporal knowledge of the sensor's coverage, independent of the on-line presence of sensor data, allows much needed search and query capabilities for determining sensor coverage for given a location or time, or for determining coincident sampling between two or more sensors.

In addition to the significant benefits discussed above, on-demand processing of geolocation has in many cases been shown to be as fast or faster than reading in and processing pre-calculated location values. With CPU power increasing faster than I/O rates, the improvement in on-demand calculation of geolocation will increase even further in the future, with the added benefit of not wasting valuable capacity with storage of blocks of latitude and longitude values.

6.4 Importance to Sensor Web Enablement

The SWE architecture and design provides an important contribution for the enablement of future sensor webs. However, it is important to note that while SensorML is a key component to the SWE initiative, SensorML is not dependent on the SWE framework and can be used on its own or in conjunction with other sensor system architectures.

In much the same way that HTML and HTTP standards enabled the exchange of any type of information on the World Wide Web, the OGC Sensor Web Enablement (SWE) initiative is focused on developing standards to enable the discovery and exchange of sensor observations, as well as the tasking of sensor systems. The functionality that has been targeted within a sensor web includes:

- Discovery of sensors and sensor observations that meet our needs
- Determination of a sensor's capabilities and quality of measurements
- Access to sensor parameters and processes that automatically allow software to process and geolocate observations
- Retrieval of real-time or time-series observations and coverages in standard encodings
- Tasking of sensors to acquire observations of interest
- Subscription to and publishing of alerts to be issued by sensors or sensor services based upon certain criteria

SensorML is a key component for enabling autonomous and intelligent sensor webs. SensorML provides the information needed for discovery of sensors, including the sensor's capabilities, location, and taskability. It also provides the means by which real-time observations can be geolocated and processed "on-the-fly" by SensorML-aware software. SensorML describes the interface and taskable parameters by which sensor tasking services can be enabled, and allows information about the sensor to accompany alerts that are published by sensor systems. Finally, intelligent sensors can utilize SensorML descriptions during on-board processing to process and determine the location of its observations.

7 Design Criteria and Assumptions for SensorML

7.1 Basic definition of a sensor

Sensors are devices for the measurement of physical quantities. A sensor measurement can be modeled as a process by which an input phenomenon is observed by the sensor at some discrete moment in time. Some measure of some property of that phenomenon is then output from the sensor. The values of the measurement are dependent on the sampling and response characteristic of that sensor, as well as on the sampling and detection methodologies. Often, either through hardware processing or subsequent processing in software, the raw observations are processed to higher-level knowledge. The SensorML model does not try to define where observation measurement and observation processing begin or end. These are both simply considered as part of the process.

There are a great variety of sensor types from simple visual thermometers to complex electron microscopes to radiometers on-board earth orbiting satellites. In some cases, sensing may be accomplished by a person rather than a device, and the result of the "measurement" may be a category rather than a numeric quantity.

Typically, sensors fall into one of two basic types. In-situ sensors measure a physical property of the medium immediately surrounding the sensor, while remote sensors measure physical properties which can be associated with features at some distance from the sensor, generally by measuring radiation reflected or emitted from an observed object. Regardless, any geometric properties described within the SensorML schema are defined within the sensor's local coordinate frame and are only related to the geospatial domain through it frame's association with the platform, mount, and their association with some geospatial reference frame. For example, to fully describe a wind profiler's wind speed and direction measurements, the height of the sensor needs to be known as that sensor could be situated on the roof of a building, mounted to a 10-meter tower, or sitting at ground-level.

7.2 Applications of SensorML

There are at least three fundamental approaches to implementing and utilizing SensorML instance documents. In the first approach, one would define a sensor system description that remains valid over a long period of time (including perhaps the future), and all time-varying parameters or other data would be considered as potential input into the system process chain. This method can support both real-time and archived observations and provides maximum flexibility for on-demand discovery and processing of observations.

The second approach is to provide a SensorML system description that is only valid over a certain period of time. For example, a data supplier might provide a satellite image derived from a scanner, and as part of the metadata for that image, would provide a SensorML system description that allows one to geolocate or further process that imagery data.

The third approach considers a SensorML instance document as instructions on how to further process existing data. In other words, a scientist could advertise and provide a particular algorithm, using a SensorML process chain definition, that can take Earth observations from a particular sensor and derive sea surface temperature or algae concentration, for example. Applying this "drag-n-drop" executable process to any segment of the observed data would allow on-demand generation of this higher-level product, as well as enable the ability to tweak parameters within the algorithm, perhaps. In additional, one could provide simply a process chain that allows geolocation of remotely sensed observation, without necessarily needing to describe the process by which the observations were made.

7.3 Sensor aggregation concepts

Detectors are typically simple devices (real or virtual) that take one input signal and generate one or more output signals. These are typically based on simple models that map the input values to output values, based on a few fixed or adjustable parameters. There are other simple processes that might exist either in hardware or software that do basic conversion of input to output. The methodology for each of these devices is fairly well defined and can be supported in software using a general model consisting of calibration curves and look-up-tables. In SensorML, these simpler atomic processes are defined as "process models", if they are pure processes with no real connection to a spatial or temporal domain, and as "components" if they are physical processing devices, such as detectors and actuators.

Sensors typically consist of several such processes that can be linked in series or parallel resulting in a mapping from input to output that is better defined as a process chain. Other chains can be conceived that take inputs and through a series of individual processes generates new output values. In SensorML, these composite processes are modeled as "process chains" if they are pure processes that don't exist in the physical domain. For most sensors of interest to the geospatial community, there is a need to relate the sensor observations to some temporal and spatial domain in order for them to be relevant. In SensorML, composite processes that can be related in space and time are modeled as a "system". Thus, typically, a System defines a collection of sensors that are related spatially and temporally to one another, as well as related to some geospatial domain. Process chains and systems in SensorML use the Composite Design pattern, such that a process chain is itself a process, such that it can have inputs, outputs, and parameters, and can itself participate as part of a bigger process chain. [PAT1995]

7.4 Relationship of the sensor to a platform

A sensor system is typically composed of one or more sensors mounted on a platform. Only the sensor element is viewed as being able to measure physical quantities. A platform such as an aircraft (carrying a frame camera) may be able to determine its own instantaneous orientation and position, and in such a case, these measurements would be obtained by other sensors attached to the platform. Within SensorML, platforms and sensors are treated as separate entities. Typically a sensor will be modeled as consisting

of one or more detectors, whereas a platform will be modeled as a System that contains all of the sensors and defines positional and temporal relationships among them.

The platform may be staionary or moving with respect to the geodetic reference frame. For moving platforms, the geospatial position and orientation of a sensor is often derived from the platform on which it is mounted. It is therefore often beneficial to relate the sensor's coordinate frame to the coordinate frame of the platform's (e.g., through mounting angles and position) and then depend on the platform for determining geospatial positioning. It is also possible to ignore the platform's reference frame, and to strictly provide positional information relative to other sensors that provide positional observations (e.g., a GPS or IMU device). All of the frames of reference in SensorML can in general be dynamic or static. Figure 7.1 illustrates the relationship of a sensor's frame (in pink) that is fixed but has been translated and rotated relative to the moving spacecraft frame (in black).

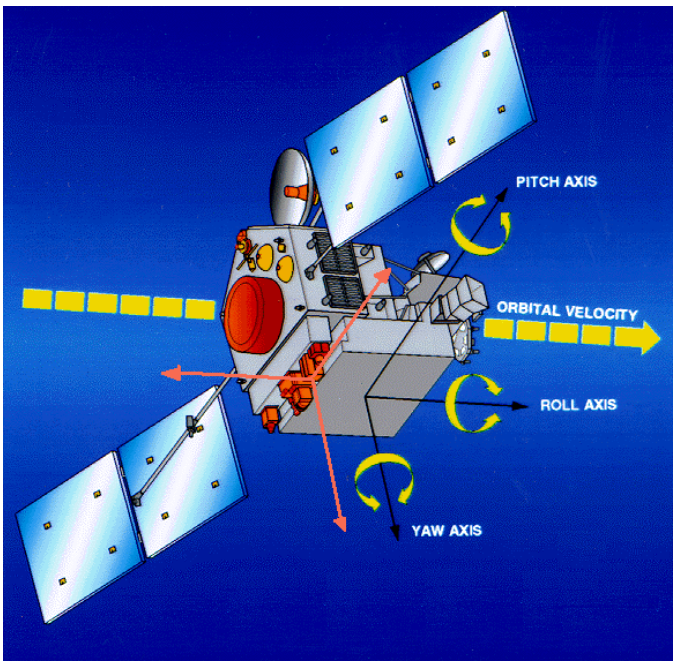


Figure 7.1. Relationship of sensor frame (pink) to the moving platform frame (black).

Based on the type of sensor and on the characteristics of the platform, a sensor system can be classified according to Table 7.1. Based on the dynamics of the platform, a sensor system may be fixed (stationary) or mobile (dynamic). Based on the sensor characteristics, a sensor system may measure either in-situ (in place) or remotely. Thus, a remote sensing atmospheric profiler might be fixed to the ground (fixed remote) or attached to an aircraft (mobile remote). Similarly, an in-situ water quality sensor might be attached to a fixed station (fixed in-situ) or to a boat (mobile in-situ).

As previously discussed, we separate the description of the sensor from that of its platform. The main importance of the associated platform(s) is in providing the

relationship of the sensor and its observations to some relevant external coordinate system (for example, a geospatial reference system).

Measures / Mobility	In-Situ	Remote
Fixed	Stationary O2 Probe	Doppler Radar station
Mobile	"Diving" Salinity probe	Airborne LIDAR

Table 7.1. Relationships between in-situ and remote sensors on mobile and fixed platforms.

In lieu of providing sensor location relative to a platform oriented coordinate reference system, one can also choose to simply relate sensor locations relative to other sensors that are responsible for providing location (e.g., a GPS sensor) and attitude (e.g., an inertial navigation system).

7.5 Coordinate reference systems

All geometric and temporal characteristics of a sensor system must be related to a specified coordinate reference system (CRS). Within the SensorML, definitions for sample geometry, look angle, and collection geometry are often described relative to the sensor's CRS. In such cases, it is only through the sensor's relationship to its mount and platform(s), that the sensor and its measurements can be related to an external CRS, such as geographic latitude and longitude.

This is accomplished by defining CRSs and describing their relationships to one another. The relationship between CRSs can be accomplished either by describing a transform process between the coordinate reference systems or by defining the state of the object relative to a CRS. For instance, an individual sample's geometry (e.g., shape and size) is defined in the localized coordinates of that sample. Its relationship to a sensor's frame may be specified through a collection geometry definition. The sensor's CRS may, in turn, be related to its platform's CRS through its mounting angles and position. Finally, the platform's CRS is related to a geospatial CRS by defining its position and orientation within that CRS. The successive transformation of each of these coordinate frames into its parent CRS provides the information necessary to georegister the sensor's measurements. It is also possible, using particular sensor models within SensorML, to relate the sample geometry and position directly to a geospatial CRS.

For a remote sensor, it is necessary to determine the intersection of a pixel's look ray and the surface of the sensor's target (e.g., the Earth's ellipsoid). Typically the look angle and the sensors target are transformed into a common spatial reference frame, such as the Earth-Centered Earth-Fixed (ECEF) or Earth Centered Inertial (ECI) reference system. For in-situ sensors, the process is typically much easier.

The CRS concept will also be applied to temporal domain when applicable. One local time frame that is useful for defining the geometry and dynamics of scanners is seconds past the start of a scan (scan start time). Also, for some sensor systems, time is recorded relative to a local clock or the start of the mission. In such cases, time frames and their relationship to "Earth time" will be defined in SensorML.

7.6 Measurement / observation concepts

A sensor is designed to measure a particular property within a given sample space. When these measurements are taken, they result in an observation that may be immediately utilized or stored. In its lowest level, this observation is typically a proxy measurement of some property other than the desired physical property, itself. For example, an observation may be the height of mercury in a thermometer or the voltage across a circuit. In order for these observations to be related to a more useful physical property, a new observation will typically be derived using known sensor calibration functions and perhaps other processing algorithms.

SensorML allows one to describe whatever level of observations the creator of the document wishes to expose. For example, one might specify that the sensor measures raw voltages and then provide calibration descriptions that would allow conversion to other physical quantities. Alternatively, or in addition to, the sensor description might specify that the sensor measures temperature, and then expose the calibration used to derive those temperature values, or not.

A SensorML document will describe what physical properties are measured by the sensor, as well as information concerning the properties and quality of these measurements. In addition, a SensorML document may provide or link to the values of these measurements using one or more data provider types. However, a SensorML document does not typically contain the observation values resulting from the measurement.

7.7 Sensor response characteristics

The response characteristics of a sensor determine how the sensor will react to a particular stimulus (i.e., phenomenon) and how it will operate under given environmental conditions. Within the sensor response characteristics will be specifications for sensitivity (e.g., threshold, dynamic range, capacity, band width, etc.), accuracy and precision, and behavior under certain environmental conditions.

A very large number of sensor response characteristics can be defined using a general response model (e.g., the detector model defined in Annex C). However, there may be specific sensors that require addition of different parameters to fully describe their response behavior. Where possible, these should be derived from the general model.

7.8 Sample and collection geometry concepts

As discussed above, a sensor measures some property within a spatially and temporally defined sample. In the case of an in-situ sensor, this sample includes some spatial volume

in the immediate vicinity of the sensor. This volume may be infinitesimally small or it may be unknown or unimportant. For remote sensors, the sample involves some volume or surface area located away from the immediate vicinity of the sensor.

The geometry of a sample may be specified relative to any coordinate system. However, particularly for a remote sensor, the geometric descriptions in SensorML are typically defined relative to the sensor's local coordinate frames and not a geospatial coordinate frame. As discussed before, this allows the same sensor model to be "attached" to any stationary or dynamic platform without a need to significantly change the SensorML description. In such a case, an individual sample's geometry, such as perhaps its size, shape, or point-spread function, is described relative to a local sample coordinate frame.

This sample frame can be related to the sensor's frame by either a simple transformation or in the case of collection of samples, by a more complex transformation involving arrays or scan patterns. Possible transformations for sample collections include unstructured grids, regular arrays, scanners, frame cameras, and mathematical functions.

8 SWE Common Conceptual Models

This document defines several basic value types and data encodings that will exist in the Sensor Web Enablement (SWE) Common namespace. These include definitions that are expected to be shared among all SWE encodings and services. It is anticipated that in future releases, these SWE Common components will be defined as a separate standard on which SensorML depends.

SWE Common provides a set of data types and related components that are required in various places across the suite of OGC SWE technologies. These fall into the following categories:

- primitive data types, complementing those implemented in GML
- general purpose aggregate data types, including records, arrays, vectors and matrices
- aggregate data types with specialized semantics, including position, curve, and time-aggregates
- standard encodings to add semantics, quality indication and constraints to both primitive and aggregate types
- specialized components to support semantic definitions, as required above
- a notation for the description of XML and non-XML array encodings.

The last item in the list relates to a particularly important use case, concerning the transport of large datasets organized as records and arrays. Such data may be represented for transport in fully XML encoded form with each data item in a separate element. However, it is often desirable to allow alternative encodings of large volumes of data for transport, for both efficiency reasons and also for compatibility with legacy systems.

8.1 Simple Data Types

Of primary importance are the definitions of primitive values. Following patterns described by Fowler [Fow1998], the scalar values *Quantity*, *Count*, *Boolean*, *Category*, and *Time* provide the basic primitives. Within SensorML, these serve as the basis for specifying all inputs, outputs, and parameters within a Process. The conceptual model for simple data types is provided in Fig. 8.1.

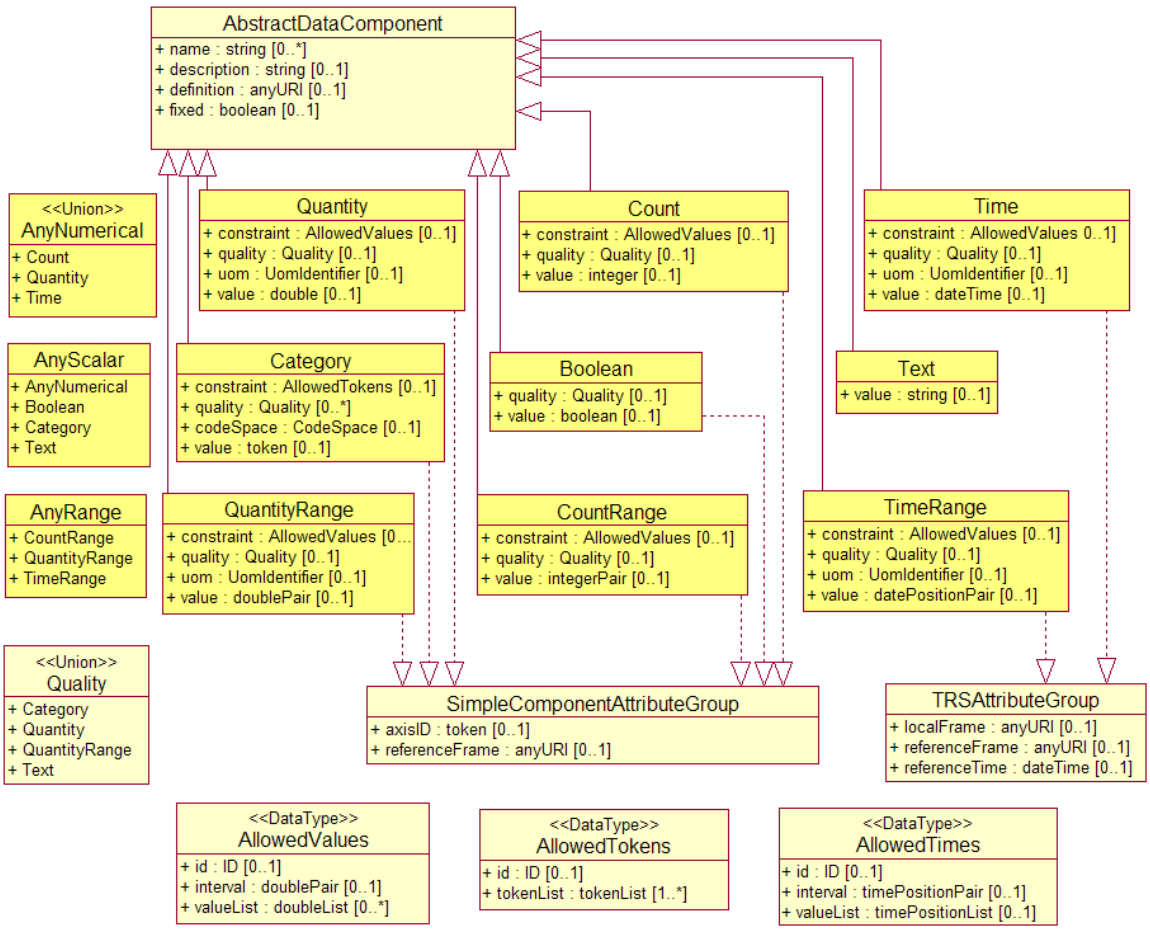


Figure 8.1. Simple Data Types Conceptual Model

8.1.1 DataComponents

A data component represents an object whose value can be assigned to a property. The simple types carry standard properties inherited from the *AbstractDataComponent* class. These include components supporting cross-referencing (inherited transitively from a class designated *AnyIdentifiableObject*), an optional link to a *definition* providing the component semantics, and an optional flag to indicate if the component value is *fixed*. The *definition* attribute identifies the phenomenon association or other context of the value and takes a definition reference as its value.

> Example: the definition may indicate that the value represents an atmospheric temperature or angle of a rotation. Its value should be obtained from a register of community-accepted terms.

As with all reference types in these conceptual diagrams, it is envisioned that these references will point to community-accepted terms that exist in online dictionaries or registries.

The *fixed* attribute, when set to true, signifies that the value cannot be changed dynamically. Data components also include name and description properties, which provide labels and

Some types also realize attributes from the *SimpleComponentAttributeGroup*. The *axisID* allows a component value to be related to the axis of a defined *referenceFrame*. Many may also be qualified by a *quality* description, and limited by *constraints* expressed as enumerated values or permitted ranges..

The *value* attribute in all of these types is optional. When it is absent the component serves as a "template" or descriptor, in a definition or schema for a data structure.

NOTE: The definition of Record in ISO 19103 provides for its schema to be provided in an associated RecordType. When used in "template" mode a SWE Common data aggregate implements RecordType. When populated with values, the SWE Common data aggregate is self-describing: the record-type information is interleaved with the record.

8.1.2 Boolean

Boolean is a simple data component that represents a property that can be true or false and takes *definition*, *axisID*, and *fixed* as attributes. The *quality* of a *Boolean* should indicate the level of confidence in the answer being true or false, expressed as a Quantity (e.g. 85%) or a Category (e.g. confidence = high).

8.1.3 Category

Category represents textual data that is a member of some larger grouping of values. The type of category is defined by the *definition* attribute (e.g., "dog" may be the value of a category defined by "mammal"), while the *codeSpace* provides a dictionary of potential values. Category usually takes a token (i.e., simple string) as its value. A *Category* can also have a *constraint* that further limits its potential value to being a member of an enumerated list of tokens. The quality of a Category would typically be a Quantity or a Category indicating the level of confidence in the classification.

8.1.4 Text

Text represents a property that takes a string as its value. Like all data components, it can be unambiguously defined by the *definition* attribute.

8.1.5 Numerical data types

Numerical data types are simple data components that take one or more numbers as its value. In addition to attributes inherited from *AbstractDataComponent*, a numerical provides a unit of measure declaration through the *uom* attribute. Thus, a numerical can specify that its value should be interpreted as a kilogram or as meters per second, for instance. The unit of measure designation shall also be able to specify a multiplication scale factor. Any numerical can have a *constraint* attribute that takes either a value range or a enumerated list of values. A numerical can also allows for a measure of its *quality*, typically expressed in terms of accuracy, precision, tolerance, or confidence.

8.1.6 Quantity and QuantityRange

Quantity represents a numerical that can be quantified using a decimal value. Quantities represent a real continuous value along some defined axis, and thus are usually encoded

using a decimal number (floating-point). Similarly, a *QuantityRange* provides minimum and maximum values as a decimal pair in that order.

8.1.7 Count and CountRange

Count represents a 'countable' property that can be quantified using an integer value. A Count includes many of the same attributes as *Quantity*, including *definition*, *axisID*, *constraint*, and *fixed,* which have the same meaning. A *CountRange* provides minimum and maximum values as an integer pair in that order.

> Example: *Count* can be used to represent a number of persons, a number of pixels, etc., and is particularly useful to define array indices.

8.1.8 Time and TimeRange

Time is supported and treated as a special type of numerical scalar, to which it adds the attributes *referenceFrame* and *localFrame*. These are used to identify the frame of reference (i.e., frame which the time value is given relative to or epoch) and the frame of interest (i.e., frame of which we are giving the time origin).

> Example: in the case of a scanner instrument, this can be useful to specify that the 'scan start time' is given relative to 'mission start time'.

One difference with a *Quantity* is that time is often expressed using a calendar based numerical system such as the ISO 8601 format. SensorML will support this possibility through the encoding of the *Time* component. The *Time* component can then be used to specify a time after an epoch (given by the *referenceTime* attribute) using a decimal value with a unit or an ISO style date and/or time. Note that the *referenceTime* should be specified even for what is commonly called an 'absolute time' since there can be confusion between GPS (or Atomic) time and International Earth Rotation and Reference Systems (IERS) time, which are typically offset by a few seconds (leap seconds). A *TimeRange* provides minimum and maximum values as a time value pair in that order.

8.1.9 Union Types

8.1.9.1 AnyNumerical, AnyScalar, AnyRange

These union types allow a choice of numeric types, of scalar types, and range types to be specified in a content model.

8.2 Aggregate Data Types

These basic data types can be grouped within any of several aggregate objects. Following conventions set in ISO 11404, the generic aggregate components supported by SensorML consists of *RecordTypes* and *ArrayTypes*. A generic record is composed of fields, while a generic array is composed of elements (ISO/IEC 11404). Derived aggregates include *DataRecord*, *SimpleDataRecord, DataArray*, *Vector*, *ConditionalValue*, and *Curve*.

8.2.1 Generic Aggregates

The conceptual models for generic data aggregates is presented in Fig. 8.2.

As with simple data types, all data aggregates provide *definition, fixed*, and *description* attributes. In this case, however, the definition provides meaning to the group as a whole rather than its individual components.

Example: a particular *DataRecord* might represent a collection of error codes coming from a GPS device. A particular *DataArray* might represent a collection of pixels that define a scan line. A particular *Vector* might define the pitch, roll, and yaw of an aircraft, while a normalized curve might define the calibration of a particular detector.

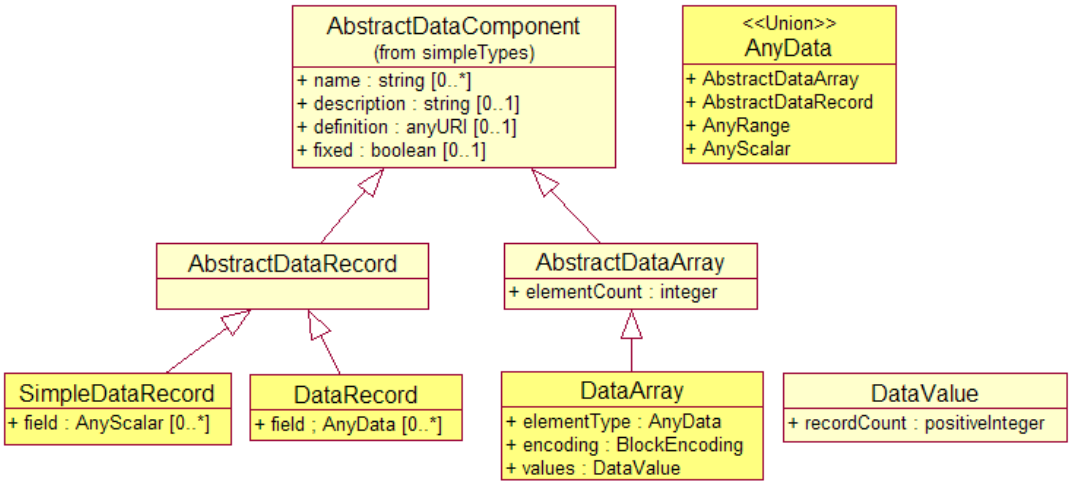


Figure 8.2. Conceptual model for Generic Data Aggregates

8.2.1.1 DataRecord

DataRecord defines some logical collection of data values of any type, and is modeled after ISO 11404. Like other data components, the *DataRecord* itself can contain *definition, fixed*, and *description* properties. The elements of a *DataRecord* are defined using the *field* property which takes *AnyData* as its value. Thus, a *DataRecord* is based on a Composite Design pattern [PAT1995], such that its *field* property can include simple data types or other aggregate data types, such as *DataRecord* or *DataArray.*

NOTE: The definition of Record in ISO 19103 provides for its schema to be provided in an associated RecordType. When used in "template" mode a SWE Common data aggregate implements RecordType. When populated with values, the SWE Common data aggregate is self-describing: the record-type information is interleaved with the record.

8.2.1.2 SimpleDataRecord

A *SimpleDataRecord* is identical to a *DataRecord* except that only allows scalar values for its *field* properties.

8.2.1.3 DataArray

DataArray is a concrete implementation of *ArrayType* base class for defining an collection of values of a *AnyDataType,* as specified by the *elementType* property. This

includes arrays of arrays and records, as well as arrays of simple data elements. Like all data components, *DataArray* includes the *definition*, *fixed*, and *description* attributes. In addition, *ArrayType* takes an *elementCount* which specifies the number of elements in the array. As described below, the values of array elements can be provided in an efficient data block which can be provided in various encodings. Encoding options provided by the optional *encoding* property will be described in Section 8.3. If the *encoding* property is omitted, then the encoding shall be assumed to be of the type *TextBlock*.

Example: *DataArray* may be used to define a simple array of *AnyScalar* values (e.g., temperature), as well as be used to define an array of *RecordType* elements, in which case, the value would be expected to be an array of like tuples (e.g., 200 occurrences of a DataRecord or Vector).

Similarly, *DataArray* allows for the nesting of arrays, since the component of a *DataArray* may itself be another DataArray. A good example of nested arrays could be an image structure. A 640x480 RGB image is an array (size 640) of an array (size 480) of a *DataRecord* containing red ,green and blue components. Other examples that should make these concepts clearer will be provided in the encoding sections to follow.

DataArray values will consist of a list of *AnySimpleType* data values with the order of the values in the list corresponding to the *elementType* definition. Furthermore, the value list can be provided in various encodings (e.g. text block, binary base64 block, or simple mime type) as will be describe more fully in Section 8.3. These will be discussed in more detail along with examples in later sections (particularly Section 10.2).

8.2.1.4 Union Type: AnyData

The *AnyData* union type allows a choice of any simple and aggregate types to be specified in a content model.

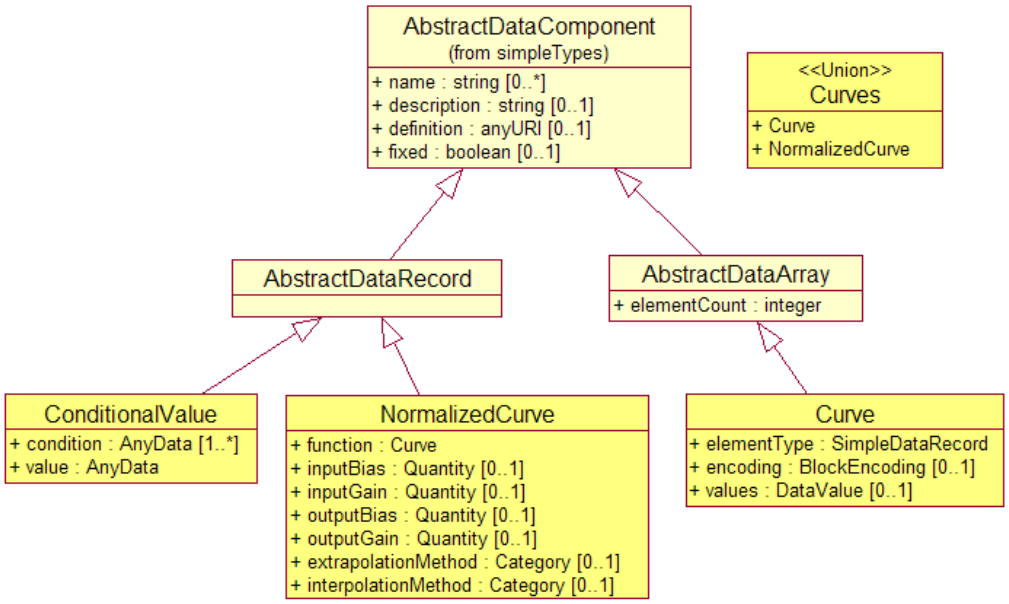


Figure 8.3. Conceptual models for Special Aggregates

8.2.2 Specialized Aggregates

Specialized aggregates are derived from AbstractDataArray and AbstractDataRecord. These are shown in Figure 8.3.

8.2.2.1 ConditionalValue

A *ConditionalValue* is a special case of a record whose *value* is dependent on one or more *condition* properties. Each *condition* can take *AnyData*, including perhaps a *Quantity* (e.g., an atmospheric pressure measured at a temperature of 25 degrees Celsius), a *Text* description of the condition (e.g., sensitivity based on a given procedure), or a *Time* (e.g., calibration curve measured at a given time).

8.2.2.2 Curve

A *Curve* is a special case of an A*rrayType* that provides an ordered set of coordinate values with respect to one independent axis. The axes of the *Curve* are defined by the *SimpleDataRecord* in the *elementType* property, with the first *field* in the *SimpleDataRecord* being considered as the independent axis, and the other fields considered to be relative to this axis. As with *DataArray*, the curve's coordinate values are provided by the *values* property whose encoding is described by the *encoding* property.

> For instance, when used inside of a process, the *Curve* provides a look-up-table by which one can map *input* values to *output* values.

8.2.2.3 NormalizedCurve

A *NormalizedCurve* allows one to further characterize a curve form using gains and biases. It also allows one to specify the intended extrapolation and interpolations methods. A *NormalizedCurve* must not have more than two Coordinates.

A *NormalizedCurve* takes a *Curve* as the value of its *function* property, but adds *inputGain*, *inputBias*, *outputGain*, and *outputBias* as additional properties. These properties alter the mapping of input values to output values according to the equation, based on the following algorithm:

> normalizedInput = input * inputGain + inputBias
>
> obtain normalizedOutput from the Curve Look-up-table based on normalizedInput
>
> output = normalizedOutput * outputGain + outputBias

The *interpolationMethod* property will define the assumed methodology to be used to interpolate between discrete values (e.g. linear or spline interpolation), while the *extrapolationMethod* property defines the assumed methodology for extending values beyond the bounds of the curve.

8.3 Position Data

Positional information is vital to the processing and utilization of sensor observations. Unlike many geospatial application that only require knowledge about location, the georegistration and processing of sensor data often requires more complete knowledge about the full state of components within the sensor system. Furthermore, the state of

many components within a sensor system is usually very dynamic, changing location and orientation with time. Thus, in SWE Common, positional information includes not only location, but can also include orientation, velocity (linear), acceleration (linear), angular velocity, and angular acceleration, as well as a time tag.

We recognize that positional data is itself the result of observation. Thus, in SWE Common, positional information is treated the same as any other data Furthermore, positional data can be output by some processes and can serve as input or parameters for others.

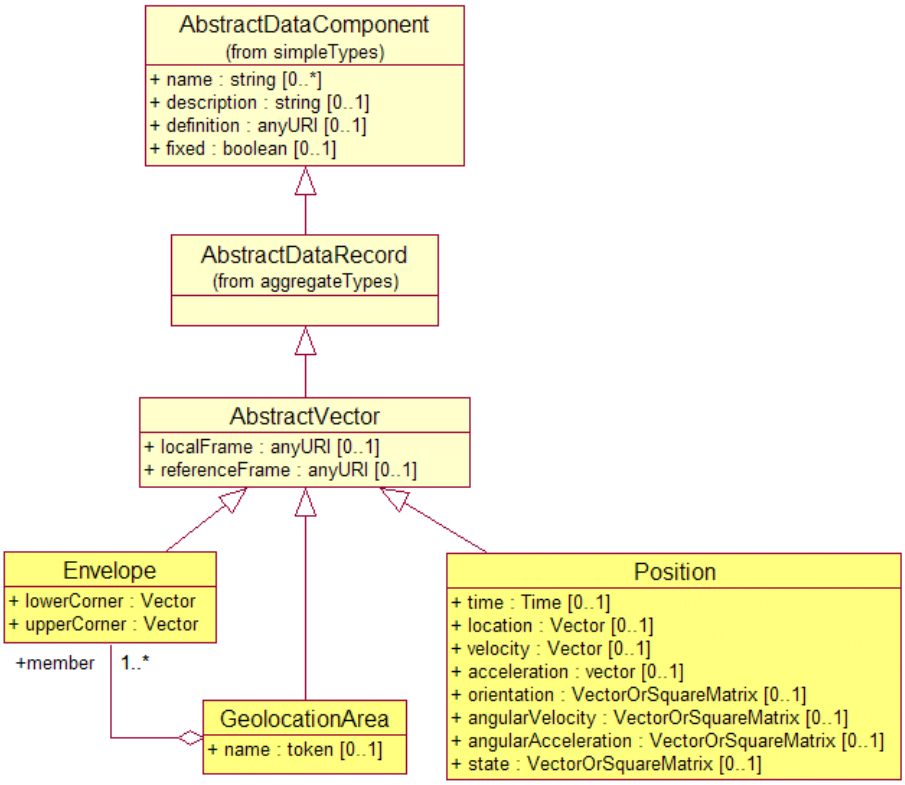


Figure 8.4. Conceptual model for Position Data

8.3.1 Position

Position is thus a special case of *RecordType* consisting of state properties and uses a *Vector* or *SquareMatrix* as the value of various state properties. *Position* can also be expressed as a single *state* property which takes a *Vector* or more likely, a *SquareMatrix* as its value.

Position requires specification of its spatial-temporal coordinate frame. In addition to geodetic coordinate frames, engineering coordinate frames may be used. Positional data is used to specify the position of a local coordinate frame to an external reference coordinate frame. These are specified by the *localFrame* and *referenceFrame* attributes, respectively. Thus, the translational properties of a Position (i.e. location, velocity, and acceleration) describe the relationship of the local frame's origin to the origin of the

reference frame. The rotational properties (i.e. orientation, angularVelocity, and AngularAcceleration) define the relationship of the local frame's axes to the equivalent axes of the reference frame. Further discussion of *Position*, along with examples, will be provided in Section 11.

8.3.2 Envelope, GeoLocationArea

An *Envelope* specifies a region within a coordinate reference frame by specifying the *lowerCorner* and *upperCorner* of the box. A *GeoLocationArea* can consist of multiple regions, each defined by its *Envelope*.

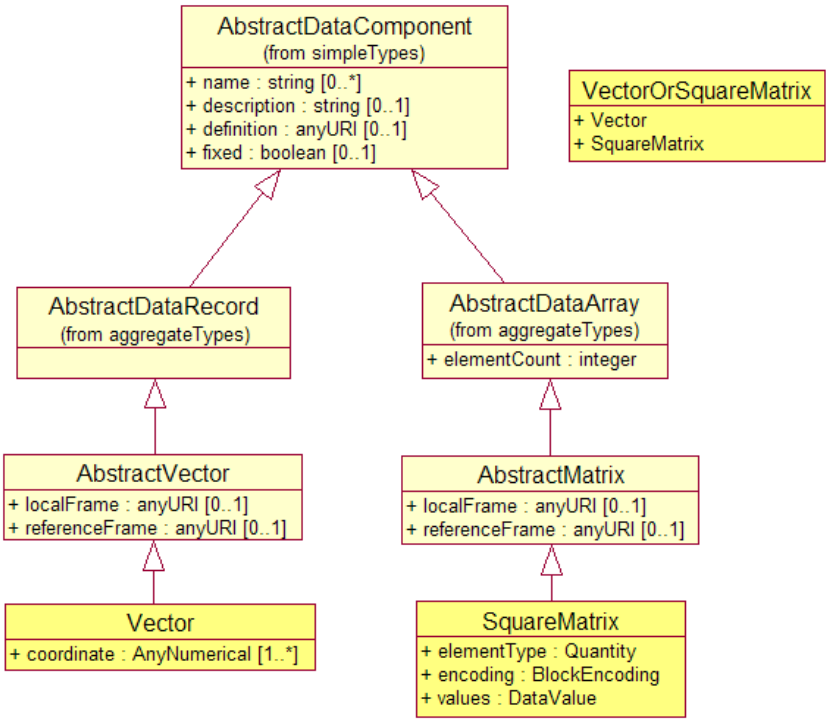


Figure 8.5. Conceptual models for Vectors and Matrices.

8.3.3 Vector

A *Vector* is a special case of a *RecordType* that takes a collection of *coordinate* properties of the type *AnyNumerical* (e.g. *Quantity*, *Count*, or *Time*). In particular, a *Vector* provides for optional *localFrame* and *referenceFrame* properties. If the coordinate frame definitions are present, the *Vector* coordinate values specify the relationship of the *localFrame* to the *referenceFrame*, based on the *definition* property of the coordinates (e.g. translation, rotation, velocity, etc).

Note: Vectors are most commonly used to describe location, orientation, velocity, and acceleration within temporal and spatial domains, but can be used to express relationships within any orthogonal frame.

8.3.4 SquareMatrix

A *SquareMatrix* is a special case of an *ArrayType* that is considered to be two-dimensional with an equal number of elements in both dimensions. *SquareMatrix* is derived from *AbstractMatrix* and thus provides for optional *localFrame* and *referenceFrame* properties. If the coordinate frame definitions are present, the *SquareMatrix* coordinate values specify the transformational relationship of the *localFrame* to the *referenceFrame*. For a *SquareMatrix*, the required *elementCount* must equal the total number of elements in the matrix.

Example: a 3x3 square matrix would have an *elementCount* of 9.

The *elementType* for a *SquareMatrix* must be of type *Quantity*. The *values* can be encoded according to the *encoding* property.

Note: A *SquareMatrix* is typically be used to define a transformation matrix.

8.4 Temporal Aggregates

8.4.1 Scope

ISO 19108 provides a basic temporal schema. However, there are several gaps in the schema which may be relevant in SWE. These particularly concern temporal aggregates, which occur as the independent axis of a time series.

ISO 19108 provides TM_TopologicComplex but the matching temporal geometric complex is missing. Furthermore, there is no class for temporal aggregates, to parallel GM_Aggregate provided in ISO 19107 Spatial Schema.

ISO 19123 provides the CV_Grid class that may be used to describe a grid in a temporal frame using time positions expressed numerically in a temporal coordinate system (e.g. GPS time). We propose that a model for grids based on the conventional temporal encodings is also required.

The model shown in Fig. 8.6 extends the ISO 19108 temporal schema to provide the required classes.

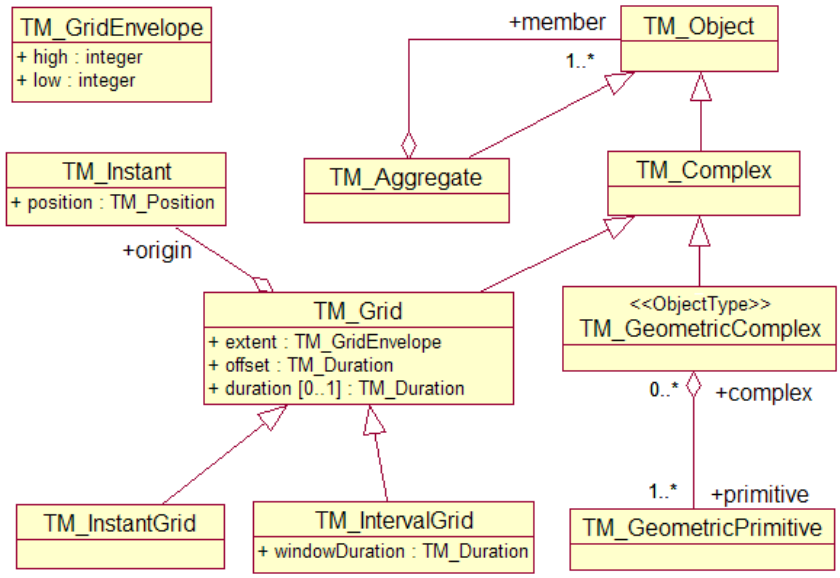


Figure 8.6. Temporal Aggregates, geometric complex, and grid

NOTE: these classes provide functionality which is also supported by the DataArray class described above, when populated with Time elements as array elements. The schema described in this sub-clause is intended to provide a model that is consistent with the idioms of ISO 19108 and ISO 19123.

8.4.2 TM_Aggregate

A *TM_Aggregate* is composed of *member* with *TM_Object* instances as values.

8.4.3 TM_GeometricComplex

A *TM_GeometricComplex* is composed of primitive *TM_GeometricPrimitive* (*TM_Instant, TM_Period*) instances.

8.4.4 TM_Grid, TM_InstantGrid, TM_IntervalGrid

The model for a temporal grid is modelled on ISO 19123 CV_Grid. A *TM_Grid* is described primarily by its *extent,* which is a pair of integers corresponding to the *low* and *high* index of the grid in the 1-D case, and the *offsetDuration* giving the grid spacing. The total temporal *duration* covered by the grid may also be recorded. The grid is tied to an external frame by its *origin* which gives the position of the grid point whose index=0.

Two concrete specializations are provided.

TM_InstantGrid is the array of time instants corresponding to the grid points.

TM_IntervalGrid is a regular array of time intervals. The duration of each interval is described by the value of the *windowDuration*. If the value of *windowDuration* is equal to the *offsetDuration*, then the grid intervals cover the sample space exactly.

8.5 Encoding

Data aggregate values can be provided in various encodings, such as a text block encoded in ASCII, a binary block (base 64 or true binary), or as a standard format defined by a MIME-type (e.g., jpeg image). The conceptual model for encoding is provided in Fig. 8.7.

8.5.1 TextBlock

The *TextBlock* encoding type is used to describe a list or stream of text values separated by specific characters, and encoded in ASCII, given in the order specified in the *dataComponents* section. It is possible to define a *tokenSeparator* which should be used between values in the data structure, as well as a *blockSeparator* which should be used to separate consecutive block of values (each block corresponds to one instance of the data structure specified in the *dataComponents* section). Both *tokenSeparator* and *blockSeparator* can be identical, but different values are useful if the transmission is error prone and needs frequent resynchronizations. The *decimalSeparator* property defines the character that separates the integer part of the decimal number from the fractional part.

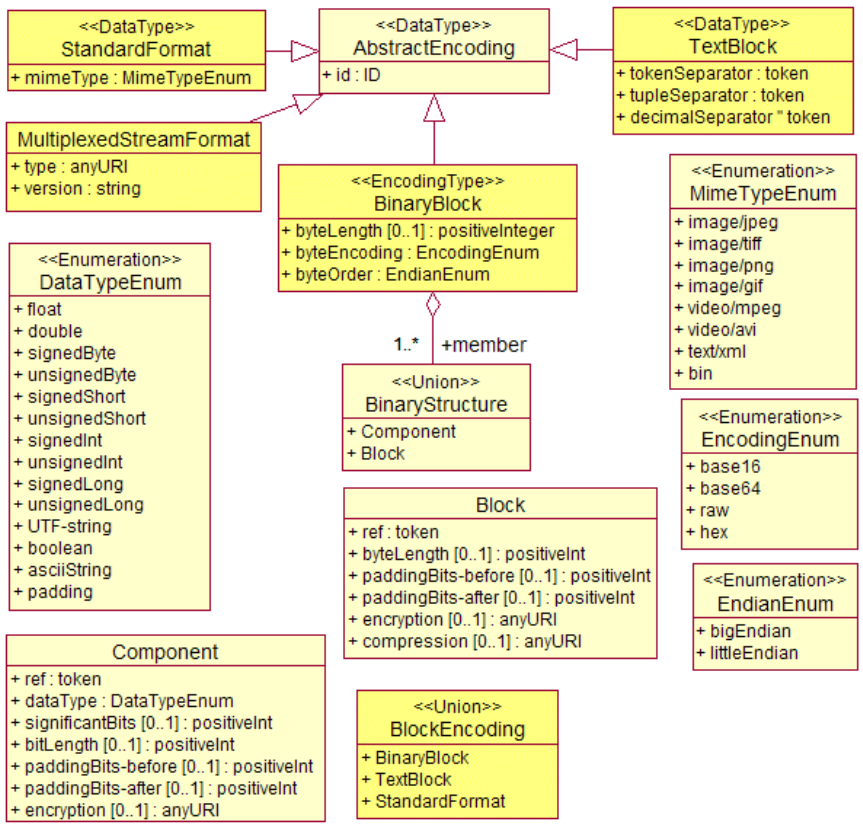


Figure 8.7. Encoding model.

8.5.2 BinaryBlock

The *BinaryBlock* encoding type is used to describe a binary data stream containing values of quantities listed in the *dataComponents* section. The *byteEncoding* attribute specifies the stream's byte encoding method, which can be raw binary if data is available in an "out of band" fashion, but also base64 or base16 (hex) in order to allow the data to be encapsulated in an XML document. The *byteOrder* attribute specifies if multiple bytes data types are written with the Most Significant Byte first (*bigEndian*) or Least Significant Byte first (*littleEndian*). The *BinaryBlock* object then takes a list of members that can be either *Component* or *Block* and which further define the parameters for the encoding of each scalar value or block of data in the data structure..

Using *Component*, each scalar value can be attributed a *dataType* (i.e., byte, short, int, long, float, double, ASCII-string, UTF-string, etc.), a number of *significantBits*, a number of *paddingBits* before and after the value, as well as *encryption* method used. Using *Block*, encoding parameters for a block of data (such as an array or group) can similarly be specified, including *byteLength*, *paddingBytes* before and after the block as well as *encryption* and *compression*. This allows for defining the compression or encryption scheme used for a block of data (e.g. specifying JPEG compression for an image). In both *Component* and *Block* classes, the *ref* attribute is used to point to the corresponding data component in the data structure defined previously.

8.5.3 StandardFormat

The *StandardFormat* encoding type allows one to encapsulate data presented in "well-known" formats such as jpeg or gif. This is done by specifying the *mimeType* of the corresponding format. Note that in this case, the MIME type applies to the whole data structure.

8.5.4 MultiplexedStreamFormat

MultiplexedStreamFormat specifies any format that supports the transmittal of disparate data clusters within an I/O stream. The format of the streaming data must be specified by the *type* property.

Examples of multiplexed data stream formats include Transducer Model Language (TML) data streams or Advanced Streaming Format (ASF).

8.5.5 XMLBlock

The set of encodings is completed by XMLBlock. The xmlElement attribute indicates the XML element that carries the data, designated using the standard xs:QName identifier.

8.6 Phenomenon

8.6.1 Scope

The SWE Common Phenomenon schema provides a model for the definition of the semantics of a value. In the context of the type library provided in SWE Common, the *definition* attribute may point to an instance of *Phenomenon* (as well as an element within an ontology). In the context of an Observation, an instance of *Phenomenon* may be used as the value of the *observedProperty* (see O&M).

A *Phenomenon* may be a physical property (such as temperature, length, etc.), a classification (such as species), frequency or count, or an existence indication.

Note: The term "Phenomenon" is sometimes used to refer to transient features, such as lightning or storms. Here it is used as an umbrella term to encompass definitions of any kind of feature property whose value is amenable to observation or estimation, including physical properties, classification axes, existence and occurrence assessments, etc.

Issue Name: [Phenomenon name change. (sc, 2007-04-13)]

Issue Description: It is anticipated that the name "Phenomenon" will be changed in the near future (e.g. possibly to "PropertyType") to better harmonize with the semantic community ontologies.

Resolution:

Issue Name: Phenomenon dictionary versus ontologies. (sc, 2007-02-03)]

Issue Description: Formal notations for knowledge representation are available (e.g. OWL [OWL]) and prototype ontologies for phenomena have been constructed using such technology (e.g. SWEET - see http://sweet.jpl.nasa.gov/ontology/property.owl).

SWEET is the most well known ontology for physical properties. However, SWEET is incomplete, and furthermore has limitations in the description of phenomena derived from more basic or atomic components, partly related to OWL's weakness in numerics.

EDCS "Attributes" [ISO/IEC 18025] is another formal dictionary of observable phenomena.

Future best practices will determine whether ontologies or XML dictionaries, or a combination of the two, will provide the best source for defining properties to be used for discovery.

Resolution:

A full schema for semantic definitions of phenomena is beyond the scope of this specification. Ultimately this rests on shared concepts that can only be described in natural language. However, the value of the observed property is a key classifier for the information reported in an observation. Thus, in order to support such classification, for use in discovery and requests, an ontology of observable phenomena must be available.

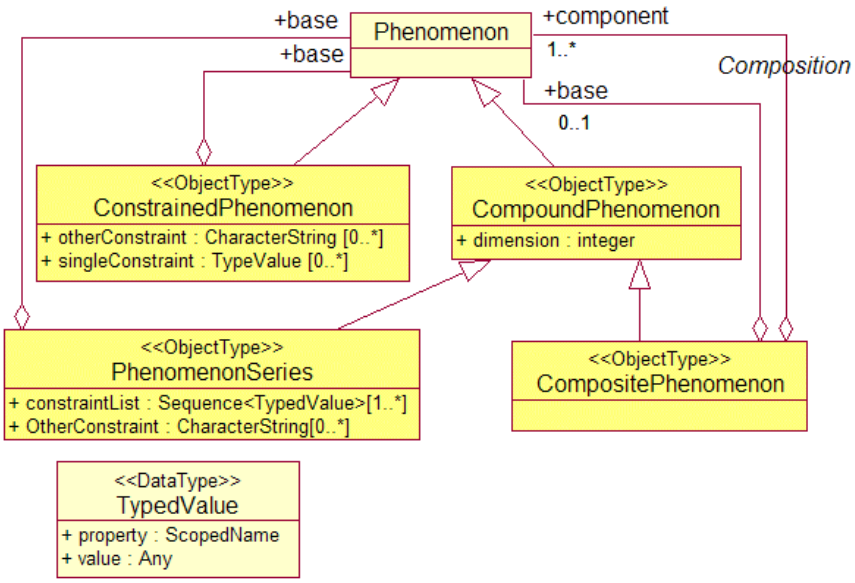


Figure 8.8. A taxonomy of phenomenon definitions

8.6.2 Derived Phenomenon

In order to support common uses in natural sciences and engineering, we provide a schema which supports the description of "derived" phenomena definitions. This requires a pre-existing set of definitions of "fundamental" phenomena to be used as the base phenomena, and for the semantics of axes of constraint for derived phenomena.

> Note: The set of "quantity types" used in the definition of units of measure (e.g. SI) may be used as basic physical properties, but this does not exhaust the base phenomena required to characterize observations.

A model for derived phenomena definitions is shown in Fig. 8.8. This may be considered primarily as a description of *requirements* for a phenomenon ontology, though it may also be implemented as a GML Application Schema following the usual encoding rules.

The description of the phenomenon will often be persisted in a dictionary (e.g. an ISO 19135 Register) and its identifier or designator used as the value in the observation instance. The phenomenon class effectively instantiates GF_PropertyType (ISO 19109).

8.6.3 Concrete Phenomenon classes

8.6.3.1 Phenomenon

The basic *Phenomenon* class is a definition, with an identifier and an optional set of aliases. Two kinds of specializations are supported: constraints and compounding.

8.6.3.2 ConstrainedPhenomenon

A *ConstrainedPhenomenon* modifies a *base* phenomenon by adding *singleConstraints*, each specifying a value on some secondary axis.

> Example: "water temperature" has the base "temperature" (i.e. it is a kind of temperature) constrained so that the property "substance" has the value "water". "Surface water temperature" might add another constraint that "depth" is "between 0 - 0.3m".

8.6.3.3 CompoundPhenomenon, CompositePhenomenon, PhenomenonSeries

A *CompoundPhenomenon* has several components, whose count is indicated by the *dimension*. *CompoundPhenomenon* is an abstract class. Two concrete specializations are provided.

A *CompositePhenomenon* is composed of a set of *component* phenomena. The components may not be related to each other, though useful compound phenomena would usually have some semantic coherence. The optional *base* phenomenon allows for the *CompositePhenomenon* to be generated by adding components to a base.

A *PhenomenonSeries* applies one or more *constraintLists* to the base phenomenon, each providing a set of values for a single secondary axis.

Example: A "radiance spectrum" may be based on "radiance" with a list of "wavelength" intervals specified.

The "base" association indicates a conceptual relationship, which may be useful in classification of observation types. The value of a specialised phenomenon must be described using a scale (units of measure, vocabulary) that could also be used for the base.

Example: an application may choose to include observations of "WaterTemperature" when the subject of interest is observations of "Temperature".

8.7 ObservableProperty

An *ObservableProperty* is a property of a phenomenon that can be observed and measured (e.g. temperature, gravitational force, chemical concentration, orientation, number-of-individuals), or a characteristic of one or more feature types, the value for which must be estimated by application of some procedure in an observation.

Example: The ObservableProperty element allows one to reference a measurable property of a phenomenon for sensor inputs or actuator outputs, for example.

In *ObservableProperty* the phenomenon property can only be defined by reference using the *definition* attribute. The *definition* attribute value should reference a property defined within a *Phenomenon* dictionary or within an ontology. An ObservableProperty can also include a name and a description.

Note: Unlike the simple data types, an ObservableProperty does NOT include properties such as uom or constraints, since these are typically characteristics of the measuring procedure and not properties of the observable phenomenon itself.

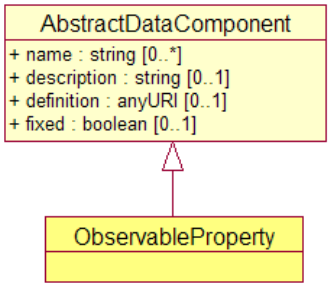


Figure 8.9. Conceptual model of ObservableProperty.

SensorML Conceptual Models

In SensorML, all components are modeled as processes. This includes components normally viewed as hardware, including transducers, actuators, and processors (which are viewed as process components) and sensors and platforms (which are modeled as systems). All components are modeled as processes that take input, and which through the application of an algorithm defined by a method and parameter values, generate output. All such components can therefore participate in process chains. Process chains are themselves processes with inputs, outputs, and parameters.

Hence, SensorML can be viewed as a specialized process description language with an emphasis on application to sensor data. Process descriptions in SensorML are agnostic of the development environment in which they might be executed, or the protocol by which data is transported between process execution modules. SensorML does not try to replace other existing technologies (such as BPEL or MATLAB Simulink). It is also conceived that SensorML-defined processes could be imported and executed within other execution environments, such as BPEL or MATLAB Simulink, as well as within SenosrML-enabled process execution software.

The models for SensorML and SWE Common are presented using Unified Model Language (UML) static structure diagrams. These UML diagrams represent conceptual models only and are not intended for automatic encoding within XML Schema.

SensorML models sensor systems as a collection of physical and non-physical processes. It also supports the description of processes that have been applied to an observation (i.e. observation lineage) or can be applied on an observation (i.e. on-demand processing). In this sense, processes in SensorML are intended as serializations of executable components. An instance of a process in SensorML should describe the inputs and outputs expected, as well as the parameters and methodology required to create output values from input values. Process models and chains are expected to be linkable, and these links are expected to be described within a process chain or system.

Issue Name: Use of term “Operation” instead of “Process”. (aw, 2006-12-07)]

Issue Description: It has been recognized that the term “Process” used in SensorML may be equivalent or similar to the use of the term “Operation” defined in ISO TC211. The issue is whether the term “Process” should be replaced with the term “Operation” within SensorML.

Resolution: This will be given serious consideration as SensorML moves toward more complete harmonization with ISO standards in future versions.

8.8 ProcessType

All processes in SensorML are derived from *AbstractProcess* which is itself derived from Abstract*Feature*. All features have as a minimum, *name* and *description* properties. In addition, all processes include *inputs*, *outputs*, and *parameters*. These three properties

take values of *AnyData* type (Fig. 9.1). The use of these SWE common data types throughout the SWE framework of encodings and services, supports linking between SWE components as well as providing a high degree of consistency.

The properties *inputs* and *outputs* represent "ports" where outside processes exchange data with this process. However, in SensorML, any *parameter* elements that do not have the property *fixed* equal to true, can also be considered as a potential input into the process. In other words, the value of any variable *parameter* within a process can be set or affected by outside processes or data sources.

In addition to *inputs*, *outputs*, and *parameters*, all processes in SensorML can provide additional information that, for the sake of clarity, have been collected within a *metadataGroup* (Section 9.4). These data, while important for resource discovery, for qualification of results, and for assistance to humans, are not considered essential to the execution of the process within a process chain. All data or information required for actual execution of the process should be included within the *inputs*, *outputs*, and *parameters* properties. It is possible, and encouraged that those parameters desired for resource discovery be linked to or copied within the appropriate metadata section.

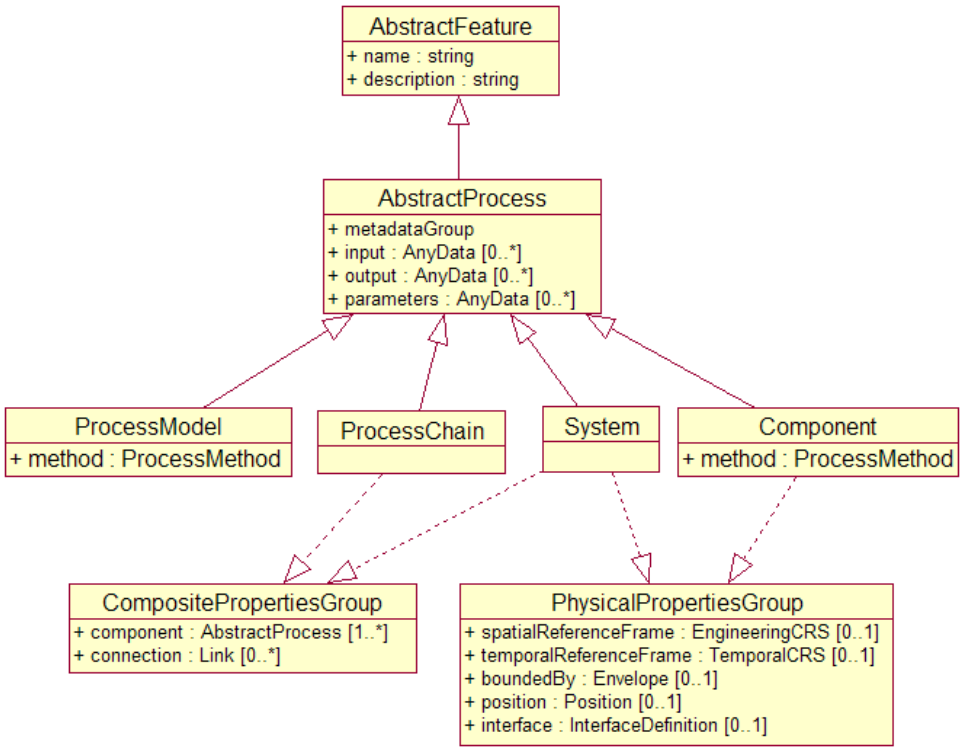


Figure 9.1. Conceptual model for Processes

As will be discussed in the following sections, all composite processes, including *ProcessChain* and *System*, realize the *CompositePropertiesGroup*, which includes the *process* and *connection* properties. These allow a description of the individual processes within the process collection and the links between them. Furthermore, processes that are

physical in nature, can include additional properties providing location information and physical interface description, as provided by realizing the *PhysicalPropertiesGroup*. These properties include *spatialReferenceFrame*, *temporalReferenceFrame*, *boundedBy*, *position*, and *interface*. Figure 9.2 provides an alternative view of processes clearing delineating between physical and non-physical processes, and atomic and composite processes.

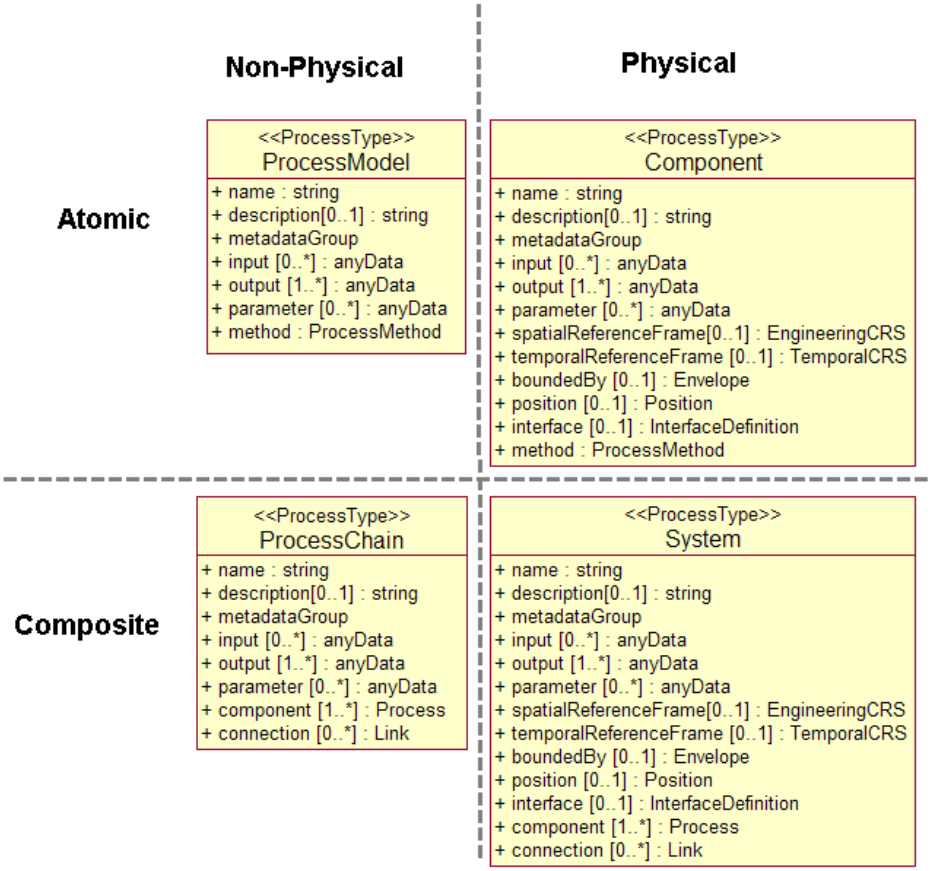


Figure 9.2. Alternative model view for physical and non-physical processes.

8.9 Non-physical (or pure) processes

Processes in SensorML are conceptually divided into two types: (1) those that are physical processes, such as detectors, actuators, and sensor systems, where information regarding their positions and interfaces may be relevant, and (2) non-physical or "pure" processes which can be treated as merely mathematical operations. Non-physical process models are shown on the left side of Fig. 9.1, while physical process models are shown on the right.

8.9.1 ProcessModel

ProcessModel is used to define more or less atomic pure processes that are expected to be used within more complex process chains. *ProcessModel* derives all properties from the base class *Process*, and adds a description of the process methodology through the *method* property. It is through execution of implementations specified within the method definition that on-demand processing using SensorML is expected to be enabled.

8.9.2 ProcessMethod

The *method* property provides the methodology by which one transforms input values to appropriate output values, based on the provided parameter values. The desire is that SensorML instances will define process chains that are self contained and executable without requiring a priori knowledge of complex processes. For this goal to be realized, SensorML-enabled software must understand how to execute the individual process models within a chain and, of course, support the flow of data between these process models according to the link defined within the process instance.

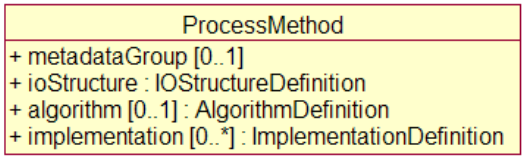


Figure 9.3. Conceptual model for a process method.

A *ProcessMethod* definition includes the metadata properties as specified more fully in Section 11.3, as well as an *IOStructureDefinition* which defines required and optional input, output, and parameter types, an *AlgorithmDefinition* which unambiguously specifies the algorithm that can be implemented (in software) or has been enabled (in hardware) for this process, and the *ImplementationDefinition* which provides information regarding executable implementations for this process.

8.9.3 ProcessChain

ProcessChain defines a collection of processes that are executable in a sequential manner to obtain a desired result. Like all processes in SensorML, *ProcessChain* itself has inputs, outputs, and parameters properties. The *inputs* and *outputs* of a *ProcessChain,* in essence, define the beginning and ending data components of the process chain, respectively. Only data components exposed through the *ProcessChain inputs*, *outputs*, or *parameters* properties are available for linking with outside processes. The *ProcessChain* also supports the *metadataGroup* properties of the base class, *Process*.

ProcessChain is based on a Composite Design pattern and thus, while it consists of a collection of other processes, it itself is a process that can participate as a component within other process chains. In addition to having all the properties of a *Process*, including *inputs*, *outputs*, *parameters*, and *metadataGroup*, the *ProcessChain* adds the zero or more *component* properties that takes *Process* (both physical and non-physical) as

their value, as well as zero to many *connection* properties that specifies the links between process components within the chain.

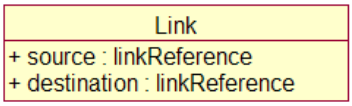


Figure 9.4. Conceptual model for a connection *Link*.

The process chain is defined by describing all of the *processes* in the chain and then describing the appropriate *connections* between components. The *connection* property includes a collection of *Link* objects that provide references to the *source* and *destination* "ports" within the various component processes. The *source* is usually an *output* from a process component or a data component within a source data object (e.g. an observation). The *destination* data component is typically an *input* or *parameter* of a process, or the output of the *ProcessChain* itself. The specifics regarding how a *linkReference* is defined will be described in the encoding Section 11.

8.10 Physical Processes

In addition to describing purely mathematical processes, SensorML describes processes that have some relationship to space and time. These might include transducers (detectors and actuators), sensor systems, samplers, and sensor platforms, for example. For much of the processing of sensor data, it is important that temporal and spatial reference frames be described and that the relationships between various reference frames be able to be explicitly defined.

All physical processes derive from the abstract *PhysicalProcess*, which has *spatialReferenceFrame* and *temporalReferenceFrame* properties that define any spatial or temporal Coordinate Reference Systems (CRS) that might be used for interrelating internal components within the process, as well as reference frames that might allow this component to be related to other components within a process chain or system. A CRS definition will include a definition of its axes, as well as the location of the CRS origin and the orientation of the axes relative to object's physical form.

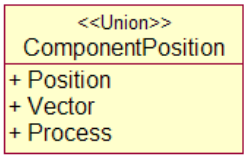


Figure 9.5. Conceptual model for an ComponentPosition.

A physical process is also modeled as having a *position* property which relates the object (as defined by its *spatialReferenceFrame*) to some external CRS. This position can be provided using *Position* or *Vector* as defined in SWE Common namespace, or through a

process (e.g. *ProcessChain*, *ProcessModel*, *Component*, or *System*) which can be used to compute dynamic position values. Additionally, the *boundedBy* property within a physical process defines the extent of the object's location within some CRS.

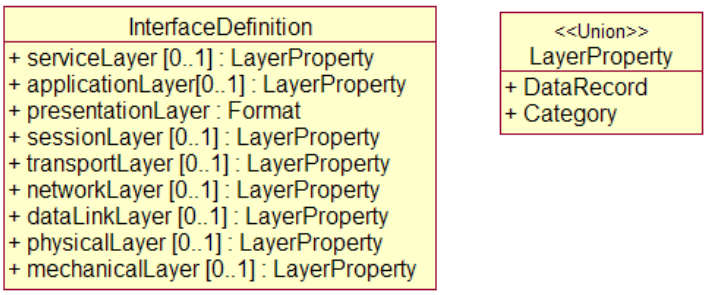


Figure 9.6. Conceptual model for an InterfaceDefinition.

Physical processes may also have zero or more interfaces over which commands and data can flow based on particular physical connections and particular protocols. The *interface* property will contain an *InterfaceDefinition* which is based on the Open Systems Interconnection (OSI) Reference Model network protocol layer model, as modeled in Fig. 9.5. Starting at the lowest-level interface, the *InterfaceDefinition* can include:

- Mechanical layer (not in OSI stack) – type of connector used (e.g. DB9, DB25, RJ45, USB-B)
- Physical layer (OSI layer 1) – provides electrical and physical characteristics of the connection including pin layouts, voltages, etc. (e.g. RS232, DSL, 802.11g)
- DataLink layers (OSI layer 2) – provides functional and procedural means of transferring data between network entities and detecting/correcting errors (e.g. Ethernet, 802.11, Token ring)
- Network layer (OSI layer 3) – provides functional and procedural means of transferring data from one source to destination via one or more networks while maintaining the Quality of Service (e.g. IP, ICMP, IPSec)
- Transport layer (OSI layer 4) – Provides transparent transfer of data between end users and can control reliability of a given link (e.g. TCP, UDP, SPX)
- Session layer (OSI layer 5) – controls the dialogues (sessions) between computers by establishing, managing, and terminating connections between the local and remote applications (e.g. NetBios, TCP session establishment)
- Presentation layer (OSI layer 6) – transforms the data to provide a standard interface for the application layer (e.g. SSL, TLS, ASCII, MIDI, MPEG, SWE Common)
- Application layer (OSI layer 7) – provides a means for the user to access information on the network through an application (e.g. HTTP, SMTP, FTP, XMPP, RTP)

- Service layer (not in OSI) – type of web service used to access data (e.g. SOS, WCS, WFS)

Each *LayerProperty* takes a *DataRecord* or *Category* as its value, with the *definition* attribute used to specify the type of protocol used (for example, RS232, USB2, etc.). The *Category* or *DataRecord* components should contain parameters meaningful for this particular protocol (e.g.. parity). The *presentationLayer* can also be specified using encoding types defined earlier in the SWE Common section.

8.10.1 Component

Any physical process can be modeled as a *Component* in SensorML if it either cannot be subdivided into smaller subprocesses, or if one chooses to treat it as a single indivisible process. A *Component* can be considered as a real-world equivalent of a *ProcessModel*. A *Component* can participate as part of a *ProcessChain* or *System*.

Component includes all the location and interface properties of any physical process and adds a *method* property that can describe the basis of physical processing of the component. In some cases, it may also be possible and desirable to provide within the *method* definition, a software implementation that can be used for simulation of the hardware component.

8.10.2 System

System is a physical equivalent of a *ProcessChain*. A System may include several physical and non-physical processes that all act to provide a certain set of *System outputs*, based on the *System inputs* and *parameters*. An example might be a weather station that includes several physical sensors as well as some on-board processing (for calculation of wind chill, for example), that all work together to provide a measure of the state of the atmosphere at particular time intervals. Similarly, an airborne remote sensing system may include, for example, the radiometric scanner (itself perhaps modeled as a system), as well as a GPS sensor and Inertial Momentum Unit for reporting location and orientation of the platform.

Being derived from *PhysicalProcess*, *System* inherits all of the properties related to location and physical interface. Like *ProcessChain*, System provides a list of processes and the links between these processes, using the *components* and *connections* properties, respectively. In addition, *System* can provide relative positions of its components through the *positions* property.

As with *Component*, *System* positions can be provided using *Position* or *Vector* as defined in SWE Common namespace or through a process (e.g. *ProcessChain*, *Component*, or *System*) used to compute dynamic position values. This position information allows derivation of the exact spatial and temporal position of measured phenomena. Each system component as well as the system itself is attached to a mathematical reference frame making the relative position unambiguous. The position list is especially important when dealing with remote sensors for which deriving the precise location of the measured phenomenon is fundamental.

8.11 Process Metadata Group

All processes and process method descriptions in SensorML provide five optional groups of metadata through the *metadataGroup*.. As previously discussed, these metadata are primarily to support discovery of resources, qualification of process results, and assistance to humans. These metadata include identifiers, classifiers, constraints, capabilities, properties, contacts, documentation sources, and history. Each of these groups will be discussed in detail below.

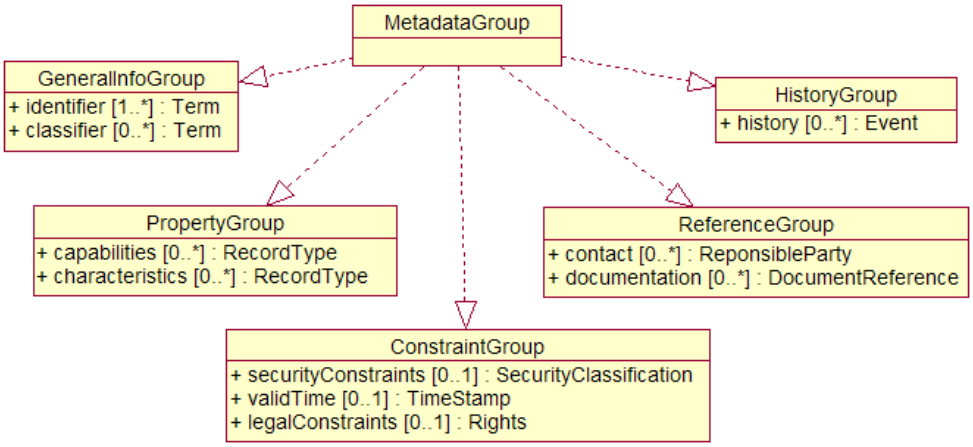


Figure 9.7. Conceptual model for the metadata group in SensorML

8.11.1 General Information (Identifier and Classifier)

The *generalInfo* group includes three properties, *identifier*, *classifier*, and *description*.

8.11.1.1 identifier

The *identifier* property take a *Term* as its value. The *Term* has a *definition* attribute that specifies in this case the type of identifier, while the *codeSpace* attribute specifies that the value of the identifier is according to the rules or enumerations of a particular authority. For example, an identifier with a *definition* of "urn:ogc:def:identication:tailNumber" might take "N291PV" as its *value* based on the *codespace* of a US Air Force rules dictionary. Other possible definitions for identifiers might include, for example, shortName, longName, acronym, serialNumber, manufacturerID, or partNumber. The *identification* properties should be considered as information suitable for the discovery process.

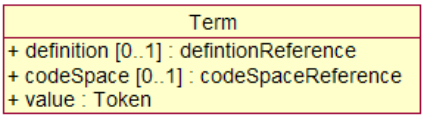


Figure 9.8. Conceptual model for Term used in identifier and classifier properties.

8.11.1.2 classification

The classification property provides a list of possible classifiers that might aid in the rapid discovery of processes, sensors, or sensor systems. Definitions for a classifier *Term* might include, for instance, sensorType, observableType, processType, intendedApplication, or missionID. The *classification* properties should be considered as information suitable for the discovery process.

8.11.2 Constraints

A SensorML resource description can be constrained by three properties: national and international *securityConstraints*, *validTime*, and *legalConstraints*. These *constraints* may or may not be considered as information suitable for the discovery process.

8.11.2.1 securityConstraints

The model for specification of security constraints may be based on such security definitions as the Security Banner Marking model of the Intelligence Community Information Security Marking (IC ISM) Standard.

8.11.2.2 validTime

The *validTime* property indicates the time instance or time range over which this process description is valid. Time constraints are important for processes in which parameter values or operation modes may change with time.

8.11.2.3 legalConstraints

The *legalConstraints* property is based on ISO 19115 and specifies whether Privacy Act, Intellectual Property Rights, or copyrights apply to the content of the process description or its use. In addition to the Boolean attributes, *privacyAct*, *intellectualPropertyRights*, and *copyrights*, *legalConstraints* take *documentation* as its value, thereby providing for more explicit description of the legal constraints.

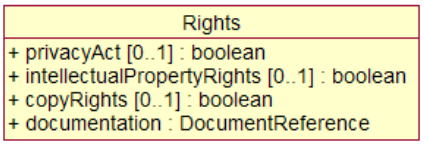


Figure 9.9. Model for *Rights* definitions used in *legalConstraints*.

8.11.3 Properties (Capabilities and Characteristics)

Any SensorML resource (e.g., a process, sensor, sensor system) may possess various *characteristics* or *capabilities* that are useful for its discovery. The *characteristics* and *capabilities* properties take a *RecordType* as their value, which allows for the grouping of various properties using SWE Common *DataRecord*, for example.

8.11.3.1 capabilities

The *capabilities* property is intended for the definition of parameters that further qualify the output of the process, component, or system for the purpose of discovery. For example, a particular remote sensor on a satellite might measure radiation between a certain spectral range (e.g. 700 to 900 nanometers) at a particular ground resolution (e.g. 5 meter), and with a typical spatial repeat period (e.g. 3.25 – 4.3 days). Alternatively, a particular process might have certain quality constraints. Any process may have certain limits (e.g., operational and survivable limits), based on physical or mathematical conditions. Once a user has identified candidate sensors based on the classifiers described above, the capabilities parameters might prove useful for further filtering of processes or sensor system during this discovery process. The *capabilities* properties should be considered as information suitable for the discovery process.

8.11.3.2 characteristics

A physical or non-physical process may have *characteristics* that may not directly qualify the output. For example, a component may have certain physical measurements such as dimensions and weight, and be constructed of a particular material. A component may have particular power demands, or anticipated lifetime. The *characteristics* properties may or may not be considered as information suitable for the discovery process.

8.11.4 References (Contacts and Documentation)

The *references* group provides *contact* and *documentation* properties that are useful for human consideration. The contact property takes two possible values for contact information: *Person*, which is based on the IC Department of Defense Discovery Metadata Specification (DDMS), and *ResponsibleParty*, which is based on ISO 19115. The documentation property provides a reference description and URI to an online resource (e.g., specification documentation, peer-reviewed algorithm literature, etc.).

8.11.5 History

Within SensorML, the history of a resource can be provided through a collection of *Event* objects. These are provided within an *EventList* that serves as the value of the *history* property. Events might for instance, specify calibration or maintenance history of a sensor, or changes to an algorithm within a process.

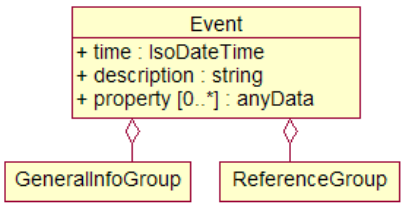


Figure 9.10. Conceptual model for *Event*.

8.12 SensorML as Applied to Sensors

The choice to model sensors and sensor systems as a process is in recognition that it is very difficult at times to decide where sample collection and detection end and where processing begins (a good example is a typical GPS sensor). It is also logical that any transducer, whether a detector, actuator, or filter, merely acts as a process that converts an input to an output. For a detector, the input is typically some physical phenomenon (e.g., temperature, position, electrical current, etc.) and the output is either some other physical phenomenon (e.g., voltage) or some digital number (DN) representing a quantification of the input phenomenon. Likewise, an actuator takes some physical phenomenon or digital signal as input and outputs a phenomenon (e.g., motion, sound, temperature, etc.) as reaction.

The SensorML model therefore also recognizes that observed data can be the result of a measurement, a simulation, or data processing chain, and thus makes no distinction between these data or the description of the processes that created them.

8.12.1 Sensor Response and Geolocation

It is further recognized that most sensor observations consist of at least two processes: a sampling process and a conversion process. The sampling process determines the subset of the surrounding environment on which a measurement is taken. For simple in-situ sensors, the sampling may be based solely on the proximity of the environment element to the detector. For a microphone and for remote sensors of radiation, the sampling process may consist of a combination of focusing devices (e.g., audio cones and baffles or optical lenses and mirrors) and on the properties of the transmitting media. Within the remote sensing community, the models for describing the collection or sampling process are typically referred to as “Sensor Models” or “Geolocation Models” and are crucial to determining the location of the observation within a geospatial domain. These Sensor Models can either be rigorous models that define the relative spatial-temporal transforms between sensor system components, or they can be functional models that define the mapping of observation elements to geospatial location based, perhaps, on polynomial equations.

The conversion process describes the response of the sensor to the stimulating phenomenon once the sample has reached the sensor’s detector. This process has typically been referred to as the “Response Model”, with calibration curves being a fundamental part of mapping the detectors input to output. Whether one chooses to treat the sampling and detection processes as separate processes or as one, is an implementation choice.

8.12.2 Observations and Data Encoding

In SensorML, the data components that result from measurements or processing from a sensor, sensor system, simulation, or process chain are specified as part of the *outputs* property. Thus, a sensor system might describe that it outputs time, latitude, longitude, altitude, temperature, and pressure, and perhaps state the appropriate units of measure. However, SensorML is not intended to provide the framework for encoding the actual

observation values. Within the SWE framework, the actual observation results should be encoded and transmitted within an O&M Observation instance or as a TML data stream.

8.12.3 Sensor Response Model

Any given sensor will include one or more detectors which convert input phenomenon to an output phenomenon or digital values. A simple detector can typically be defined as a *ProcessModel* in which its *parameters* define the response characteristics of the detector. The response model, and thus the parameters, may be as simple as a linear steady state response (e.g., calibration) curve, or may include a host of parameters such as non-linear steady state response, latency time, integration time, accuracy, impulse response, and frequency response.

This document defines a recommended (i.e., non-normative) encoding for transducers that should be suitable as a *Component* for a large number of detectors and actuators. It is restricted to one scalar input and one scalar output and uses the parameters listed above to define the transducer response. More complex transducer arrays are supported by using a *System* to wrap a *Detector* or *Actuator* description and by tweaking each response parameter using an index mechanism. It is to the advantage of the sensor community, that a general response model be developed that is common for a wide range of sensors. This would serve not only as a common response model for most transducer devices, but would also serve as a common base that one can extend to meet specific needs.

8.12.4 Sensor Models

The term “sensor model” is commonly used within the remote sensing community to refer to a model used for geolocating observations measured by a sensor located some distance from the source of the phenomenon (i.e., a remote sensor). Typically, a remote sensor detects radiation that is either reflected or transmitted by an object. The sensors that detect this radiation can utilize a complex system of lenses, filters, rotating mirrors, and various detectors, resulting in potentially complex models for determining the location of individual observations.

Remote sensors typically fall under a few categories, including frame cameras (including video cameras), profilers, and scanners (including line, conic, pushbroom, swishbroom, and volumetric scanners). In addition, several sensor models exist that utilize polynomial relationships between the location of the observation on an image, for example, and its location on a planetary surface. Such models include but are not limited to Tie-Point models, Rational Polynomial Coefficients, and Replacement Sensor Model.

The recommended (non-normative) transducer model defined in this document coupled with a *System* can be used to specify internal geometry of a sensor. Other sensor models, such as those listed above, may be derived from *Process*. As with response models, it is advantageous to the sensor community that the number of sensor models be kept to a minimum in order to assist in the development and maintenance of SensorML-enabled software capable of geolocating remotely sensed observations on-demand.

9 SWE Common XML Encoding and Examples (Informative)

The normative definition for SensorML is provided by the schema presented in Appendices A (SensorML) and B (SWE Common). The following informative section provides additional information on the SWE Common encoding through the use of XML instance "snippets" and examples.

9.1 Encoding principles

The SensorML models presented in the previous section, are encoded as XML Schema documents. It is envisioned that XML instance documents will be created for each process, component, and system based upon the framework and rules provided by the XML Schema. The current schema provides concrete elements for creating instance documents for processes, sensor components, and sensor systems and does not require the development of application schemas for various process or component types.

If it proves desirable to further restrict the schema for particular process or component types, then this can be achieved using derivation by restriction in XML Schema, by restricting the current XML Schema using Schematron rules, or through a combination of both.

9.1.1 XML Encoding Conventions

The *http://www.opengis.net/sensorML* namespace is used for fundamental core SensorML elements that can be applicable for a wide range of sensors. As they are developed, common process and component types can be approved by OGC or particular user communities and can utilize the *http://www.opengis.net/sml-x* namespace.

This document also defines schema for components used throughout the SWE framework. These are referred to as SWE Common components and are defined within the *http://www.opengis.net/swe* namespace.

The SensorML schema also utilizes components from two external XML schemas:

Geography Markup Language (GML), version 3.1.1, *http://www.opengis.net/gml*

Intelligence Community Information Security Marking, v2, *urn:us:gov:ic:ism:v2*

The "rules" used to encode the SensorML models into an XML Schema are similar to those used in GML. SensorML uses the concept that there are "Objects" and that these Objects have "properties" or "associations" that take other Objects or basic data types (e.g., string, decimal, etc.) as their values. While this convention occasionally results in deeper nesting of elements, it provides explicit association between Objects and is comparable to conventions utilized in the Resource Description Framework and the Semantic Web. SensorML follows the following lexical conventions:

- Objects are instantiated as XML elements with a conceptually meaningful name in UpperCamelCase

- Properties are instantiated as XML elements whose name is in lowerCamelCase
- Abstract elements start with an underscore (e.g. *_Process*) while element types start with the word "Abstract" (e.g. *AbstractProcessType*)
- Names of XML types are in UpperCamelCase ending in the word "Type"

As in GML, most properties and Objects in the UML diagrams are encoded as elements within the XML Schema. Only a few properties, such as IDs, xlink associations, definitions, and reference frames, are encoded as attributes within an element. Most properties that are encoded as attributes with SensorML support some form of URI links to other Objects, dictionary entries, or online references.

9.1.2 ID, URI, and Linkable Properties

As in GML, SensorML makes extensive use of XLink components to support hypertext referencing in XML. This allows one to reuse Objects that are either internal or external to the instance document, and allows these objects to perhaps have different associations or roles within one or more documents. This is supported by extensive use of the *id* attribute (taking an *xs:ID* as its value) within Objects, and utilizing the GML attribute group, *gml:AssociationGroup*, within property elements.

In properties that support XLink components, one can usually choose to define that property value inline, as in:

```
<contact xlink:arcrole="expert">
    <ResponsibleParty id="JohnDoe">
    ….
    </ResponsibleParty>
</contact>
```

Or one can reference an object within the same document:

```
<contact xlink:href="#JohnDoe" xlink:arcrole="expert"/>
```

or an object within an external document:

```
<contact xlink:href="http://www.myCom.com/personel.xml#JohnDoe"  xlink:arcrole="expert"/>
```

Typically, XLink references will be specified as URLs or as URNs that can be easily resolved through registries. In all cases above, the *xlink:arcrole* attribute was utilized to define the particular role of the contact with regard to this object.

In addition to these cases, many other attributes in SensorML are encoded to take *anyURI* as their value. SensorML is designed with the concept that there exist dictionaries or registries for definitive specification of various terms. We anticipate such definitions for phenomenon, for various community specific sensor and process properties, and for searchable terms to be used within identifiers and classifiers. Such dictionaries and registries are crucial to the goal of interoperability both within and between different sensor and processing communities. In most cases, these definitions should be specified

in SensorML documents as resolvable URNs, while XLink references will be either URLs or URNs.

9.2 SWE Common Data

SWE Common data types are used throughout SensorML and throughout other SWE encodings and web services. Within SensorML, the *AnyData* group serves as a value for *InputList*, *OutputList*, *ParameterList*, and *PropertyList*. The schema for simple data types is provided by simpleTypes.xsd within the *http://www.opengis:net/swe* namespace, while composite data types are provided within aggregateTypes.xsd, curveTypes.xsd, and positionTypes.xsd.

9.2.1 Simple data components (hard-typing and soft-typing)

In SensorML, both soft-typing and hard-typing of data parameters is supported as indicated in these soft-typed and hard-typed examples (say, within *csm* namespace):

```
<!—soft-typed -->

    <swe:field name=”fov”>
        <swe:Quantity definition="urn:ogc:def:property:OGC:focalLength" >
            <swe:uom code="mm" />
            <swe:value> 0.1 </value>
        </swe:Quantity>
    </swe:field>


<!—hard-typed -->

    <csm:fov>
        <swe:Quantity definition="urn:ogc:def:property:OGC:focalLength" >
            <swe:uom code="mm" />
            <swe:value> 0.1 </value>
        </swe:Quantity>
    </csm:fov>
```

The main difference between the two examples is that the first example takes a *xs:qname* within the *name* attribute, while the second example has the *qname* as the element name. The recommended behavior of a parser would be to utilize the *qname* within the *name* attribute if it exists and to use the element name otherwise. It is highly recommended that the hard-typing occur at the property level and not at the object level for simple data types, such as *Quantity*, *Count*, *Category*, or *Boolean*.

9.2.2 Simple data types

Elements of the *AnyScalar* group provide a rich and robust means of defining data components (e.g. inputs, outputs, and parameters) and properties (e.g. capabilities, characteristics, and interface properties). As described previous in the models, simple data types can include *Quantity*, *Category*, *Boolean*, and *Count*. Depending on the particular application, these scalar data types may or may not provide an actual value inline.

9.2.2.1 Optional data value

For example, a *Quantity* can accept either an *xs:double* as its *value* or omit the optional *value* property:

```
<swe:Quantity gml:id="elevationAngle" fixed="false" definition="urn:ogc:def:property:OGC:scanAngle">
    <uom code="deg"/>
    <swe:constraint>
        <swe:AllowedValue id="elevationAngleRange">
            <swe:interval>10.0 45.2 </swe:interval>
        </swe:AllowedValue >
    </swe:constraint>
</swe:Quantity>
```

```
<swe:Quantity gml:id="elevationAngle" fixed="false" definition="urn:ogc:def:property:OGC:scanAngle">
    <swe:uom code="deg"/>
    <swe:value> 25.3 </swe:value>
</swe:Quantity>
```

Empty components will typically be used to define components within an observation array for which values will be be provided separately. Empty components can also serve as targets of links between one or more processes or data sources within SensorML process chains or systems. *AnyData* components with the *fixed* attribute set to *true* cannot be altered through links.

9.2.2.2 Units of Measure

Quantity and *Count* objects include a *uom* property for specifying units of measure. One can specify units of measure in one of three ways. The first is to use the xlink:href attribute to reference known unit definitions that have been defined online using *gml:UnitDefinition*, as in the following example for parts-per-million.

```
<swe:uom xlink:href="urn:ogc:def:unit:OGC:ppm"/>
```

The second is to utilize the *gml:UnitDefinition* object to define the units inline. This would typically be used when one wishes to define a complex unit of measure that is perhaps not available in a standard unit dictionary, as in the following example for slugs per foot-second.

```
<gml:ConventionalUnit gml:id="slug_fts">
    <gml:name>slugs/foot-second</gml:name>
    <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
            urn:ogc:def:uom:OGC:slug_fts</gml:name>
    <gml:quantityType>dynamic viscosity</gml:quantityType>
    <gml:catalogSymbol>slug/fts</gml:catalogSymbol>
    <gml:conversionToPreferredUnit uom="#Pa.s">
        <gml:factor>47.9</gml:factor>
    </gml:conversionToPreferredUnit>
    <gml:derivationUnitTerm uom=" urn:ogc:def:unit:OGC:slug" exponent="1"/>
    <gml:derivationUnitTerm uom=" urn:ogc:def:unit:OGC:ft" exponent="-1"/>
    <gml:derivationUnitTerm uom=" urn:ogc:def:unit:OGC:s" exponent="-1"/>
</gml:ConventionalUnit>
```

The third option for specifying units of measure is to utilize the Unified Code for Units of Measure (UCUM) within the *code* attribute of *uom*, as in the following example for centimeter-squared per second [UCUM].

```
<swe:uom code="cm2.s-1"/>
```

9.2.2.3 Value constraints

The *constraints* property allows one to specify various restrictions on the possible values of the scalar. For numerical scalars such as *Quantity* and *Count*, the *AllowedValue* can take either an *interval* property as shown in the example above or a discrete *valueList* as shown in the example below.

```
<swe:Quantity gml:id="elevationAngle" fixed="false" definition="urn:ogc:def:property:OGC:scanAngle">
    <swe:uom xlink:href="urn:ogc:unit:degree"/>
    <swe:constraint>
        <swe:AllowedValue id="elevationAngleValues">
            <swe:valueList>10.0 12.7 23.5 31.9 45.2 </swe:valueList>
        </swe:AllowedValue >
    </swe:constraint>
</swe:Quantity>
```

The Category scalar type can be constrained using a list of tokens as illustrated below.

```
<swe:Category  gml:id="aerosols"definition="urn:ogc:def:property:OGC:chemicalCompounds">
    <swe:constraint>
        <swe:AllowedTokens id="chemicalSpecies">
            <swe:itemList>O2 O3 SO4 NOx </itemList>
        </swe:AllowedTokens >
    </swe:constraint>
</swe:Category >
```

9.2.2.4 Quality specification

Most of the simple scalar types have a *quality* property that can provide some measure of the quality of the scalar value. For numerical scalars such as Count, Quantity, and Time, the quality may be expressed as values of precision, accuracy, tolerance, and confidence level. In addition to the numerical scalars, *Boolean* also has an optional *quality* property which may take some measure of the confidence level as its value.

The *quality* property can take *Quantity, QuantityRange, Category,* or *Text* as its value, depending on the type of quality measure utilized. The type of quality measure should be explicitly in the *definition* attribute, as in the following example:

```
<swe:Quantity gml:id="elevationAngle" fixed="false" definition="urn:ogc:def:property:OGC:scanAngle">
    <swe:uom xlink:href="urn:ogc:unit:degree"/>
    <swe:quality>
        <swe:QuantityRange definition=" urn:ogc:def:property:OGC:tolerance2std">
            <swe:value> -0.02 0.02 </value>
        </swe:QuantityRange>
    </swe:quality>
    <swe:value> 25.3 </swe:value>
</swe:Quantity>
```

9.2.3 ObservableProperty

In *ObservableProperty* an observable phenomenon property can only be defined by reference using the *definition* attribute. The *definition* attribute value should reference a property defined within a *Phenomenon* dictionary or within an ontology.

The *ObservableProperty* element includes optional *id* and *definition* attributes, as well as optional *name* and *description* element, as in the following examples:

```
<swe:ObservableProperty gml:id="maxT" definition="urn:ogc:def:property:OGC:waterTemperature"/>

<swe:ObservableProperty definition="urn:ogc:def:property:OGC:waterTemperature"/>
    <gml:description>The temperature of the river measured at a depth of 3 meters</gml:description>
    <gml:name>Water Temperature – 3m</gml:name>
</swe:ObservableProperty>
```

9.2.4 Data Aggregates

9.2.4.1 DataRecord

Most of the properties that utilize *AnyData* within SensorML will group data components in one or more *DataRecord* objects. A *DataRecord* can itself take *id* and *definition* as attributes.

The main reason for supporting hard-typing within SensorML is to allow one to define reusable data record types using XML Schema and to explicitly specify which components are expected.

> Note: The need for hard-typing in this case is a result of limitations on how one must support rules in XML Schema; XML Schematron rules are more flexible and can allow one to specify specific groups while using soft-typing.

An example of a soft-typed data group for a GPS Status data block is provided below:

```
<!—soft-typed DataRecord-->

<output name="status">
    <swe:DataRecord definition="urn:ogc:def:property:OGC:gpsStatus">
        <swe:field name="hFOM">
            <swe:Quantity definition="urn:ogc:def:property:OGC:HFOM">
                <swe:uom code="m"/>
            </swe:Quantity>
        </swe:field>
        <swe:field name="vFOM">
            <swe:Quantity definition="urn:ogc:def:property:OGC:VFOM">
                <swe:uom code="m"/>
            </swe:Quantity>
        </swe:field>
        <swe:field name="hDOP">
            <swe:Quantity definition="urn:ogc:def:property:OGC:HDOP">
                <swe:uom code="m"/>
            </swe:Quantity>
        </swe:field >
        <swe:field name="vDOP">
            <swe:Quantity definition="urn:ogc:def:property:OGC:VDOP">
                <swe:uom code="m"/>
            </swe:Quantity>
        </swe:field>
        <swe:field name="numSys">
            <swe:Count definition="urn:ogc:def:property:OGC:NUMSYS"/>
        </swe:field >
        <swe:field name="navMode">
```

```
            <swe:Count definition="urn:ogc:def:property:OGC:NAVMODE"/>
        </swe:field>
    </swe:DataRecord>
</output>
```

A hard-typed data group will be derived from *_AbstractRecordType* by extension and will add appropriate hard-typed data component properties. An example instance from such a data group would appear as:

```
<!—hard-typed DataRecord -->

<output name="status">
    <GpsStatus>
        <hFOM>
            <swe:Quantity definition="urn:ogc:def:property:OGC:HFOM">
                <swe:uom code="m"/>
            </swe:Quantity>
        </hFOM>
        <vFOM>
            <swe:Quantity definition="urn:ogc:def:property:OGC:VFOM">
                <swe:uom code="m"/>
            </swe:Quantity>
        </vFOM>
        <hDOP>
            <swe:Quantity definition="urn:ogc:def:property:OGC:HDOP">
                <swe:uom code="m"/>
            </swe:Quantity>
        </hDOP>
        <vDOP>
            <swe:Quantity definition="urn:ogc:def:property:OGC:VDOP">
                <swe:uom code="m"/>
            </swe:Quantity>
        </vDOP>
        <numSys>
            <swe:Count definition="urn:ogc:def:property:OGC:NUMSYS"/>
        </numSys >
        <navMode>
            <swe:Count definition="urn:ogc:def:property:OGC:navMode"/>
        </navMode >
    </GpsStatus>
</output>
```

A *DataRecord* with no data values for the field, as in the above examples, may be used to define linkable *inputs*, *outputs*, or *parameters* in a process or to define data block components whose values will be provided separately. However, components within a *DataRecord* can also include values within the *value* property, as in the following example from a weather station observation.

```
<swe:DataRecord definition="urn:ogc:def:property:OGC:atmosphericConditions">
    <swe:field name="AirTemperature">
        <swe:Quantity definition="urn:ogc:def:property:OGC:AirTemperature">
                <swe:uom code="Cel"/>
                <swe:value> 35.1 </swe:value>
        </swe:Quantity>
    </swe:field>
    <swe:field name="WindSpeed">
        <swe:Quantity definition="urn:ogc:def:property:OGC:WindSpeed">
                <swe:uom code="m/s"/>
                <swe:value> 6.5 </swe:value>
        </swe:Quantity>
    </swe:field>
```

```
        <swe:field name="WindDirection">
            <swe:Quantity definition="urn:ogc:def:property:OGC:WindDirectionToNorth">
                    <swe:uom code="deg"/>
                    <swe:value> 85.0 </swe:value>
            </swe:Quantity>
        </swe:field>
        <swe:field name="AtmosphericPressure">
            <swe:Quantity definition="urn:ogc:def:property:OGC:AtmosphericPressure">
                    <swe:uom code="hPa"/>
                    <swe:value> 950.0 </swe:value>
            </swe:Quantity>
        </swe:field>
        <swe:field name="RelativeHumidity">
            <swe:Quantity definition="urn:ogc:def:property:OGC:RelativeHumidity">
                    <swe:uom code="%"/>
                    <swe:value> 32.0 </swe:value>
            </swe:Quantity>
        </swe:field>
        <swe:field name="Visibility">
            <swe:Category definition="urn:ogc:def:property:OGC:Visibility"/>
                    <swe:value> clear </swe:value>
            </swe:Category>
        </swe:field>
    </swe:DataRecord>
```

9.2.4.2 DataArray

One may also support data aggregations using an *DataArray*, as in the following example which defines the HRG scan line for the SPOT 5 satellite sensor. This example specifies that there is a 3000 member array of pixels for a single scan line and that each element (or pixel) in that array consist of a cluster of 4 components (or bands).

```
<sml:output name="HRG_scanLine">
    <swe:DataArray definition="urn:ogc:def:property:OGC:scanLine">
        <swe:elementCount>
            <swe:Count definition="urn:ogc:def:property:OGC:numberOfPixels">3000</swe:Count>
        </swe:elementCount>
        <swe:elementType name="pixel">
            <swe:DataRecord definition="urn:ogc:def:property:OGC:multiBandPixel">
                <swe:field name="xs1">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                </swe:field >
                <swe:field name="xs2">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                </swe: field >
                <swe:field name="xs3">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                </swe:field >
                <swe:field name="swir">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                </swe: field >
            </swe:DataRecord >
        </swe:elementType >
    </swe:DataArray>
</sml:output>
```

A *DataRecord* or *DataArray* with no data values for the components, as in the above example, may be used to define linkable *inputs*, *outputs*, or *parameters* in a process. However, an *DataArray* can include inline values as a compact block encoded in ASCII,

binary, or as a known simple MIME type. The following example defines an array of measurements from a weather station encoded as an ASCII block..

```
<swe:DataArray>
    <swe:elementCount>
        <swe:Count definition="urn:ogc:def:property:OGC:timeSteps">4</swe:Count>
    </swe:elementCount>
    <swe:elementType>
        <swe:DataRecord definition="urn:ogc:def:property:OGC:atmosphericConditions">
            <swe:field name="Time">
                <swe:Time definition="urn:ogc:def:property:OGC:Time">
                        <swe:uom xlink=" urn:ogc:def:unit:ISO:8601"/>
                </swe: Time>
            </swe:field>
            <swe:field name="AirTemperature">
                <swe:Quantity definition="urn:ogc:def:property:OGC:AirTemperature">
                        <swe:uom code="Cel"/>
                </swe:Quantity>
            </swe:field>
            <swe:field name="AtmosphericPressure">
                <swe:Quantity definition="urn:ogc:def:property:OGC:AtmosphericPressure">
                        <swe:uom code="hPa"/>
                </swe:Quantity>
            </swe:field>
            <swe:field name="RelativeHumidity">
                <swe:Quantity definition="urn:ogc:def:property:OGC:RelativeHumidity">
                        <swe:uom code="%"/>
                </swe:Quantity>
            </swe:field>
            <swe:field name="Visibility">
                <swe:Category definition="urn:ogc:def:property:OGC:SkyCondition"/>
            </swe:field>
        </swe:DataRecord>
    </swe:elementType>
    <swe:encoding>
        <swe:TextBlock tokenSeparator="&#x20;" blockSeparator="," decimalSeparator="."/>
    </swe:encoding>
    <swe:values>
        2006-10-05T12:30:00Z 35.1 950.0 32.0 clear,
        2006-10-05T13:00:00Z 35.8 940.0 331 clear,
        2006-10-05T13:30:00Z 36.5 938.0 35.8 hazy,
        2006-10-05T14:00:00Z 38.0 935.0 37.0 cloudy
    </swe:values>
</swe:DataArray>
```

The following example describes the scan line measurement from a remote scanner that is encoded in base64.

```
<swe:DataArray definition="urn:ogc:def:property:OGC:scanLine">
    <swe:elementCount>
        <swe:Count definition="urn:ogc:def:property:OGC:numberOfPixels">720</swe:Count>
    </swe:elementCount>
    <swe:elementType name="radiance">
        <swe:DataRecord>
            <swe:field name="band1">
                <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
            </swe:field>
            <swe:field name="band2">
                <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
            </swe:field>
            <swe:field name="band3">
                <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
            </swe:field>
            <swe:field name="band4">
                <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
```

```
            /swe:field>
        </swe:DataRecord>
    </swe:elementType>
    <swe:encoding>
        <swe:BinaryBlock byteEncoding="base64" byteOrder="bigEndian">
            <swe:member>
                <swe:Component ref="timedScanline/scanLine/radiance/band1"
                    dataType="urn:ogc:def:dataType:unsignedInteger"/>
            </swe:member>
            <swe:member>
                <swe:Component ref="timedScanline/scanLine/radiance/band2"
                    dataType="urn:ogc:def:dataType:unsignedInteger"/>
            </swe:member>
            <swe:member>
                <swe:Component ref="timedScanline/scanLine/radiance/band3"
                    dataType="urn:ogc:def:dataType:OGC:unsignedInteger"/>
            </swe:member>
            <swe:member>
                <swe:Component ref="timedScanline/scanLine/radiance/band4"
                    dataType="urn:ogc:def:dataType:OGC:unsignedInteger"/>
            </swe:member>
        </swe:BinaryBlock>
    </swe:encoding>
    <swe:values
    3BTYRdhb3OLXctMQzkzKrNMJ3YzVSM0szJ3K+NGh0afP8tSv05jUc84gzn3Xfc56zSHLzMlgyyrK
    nsrgyGvHxM1c0qnnZukb5lj5HOug8sXvRufY8tv1E+2W3+Lwb/3PCBPu4OIa7lvgBu85B/H+AOfA
    8EXtg+NZ8HnmUfA8/8fxnupj5CDu0vN76FLvovKo7QYBEgGS/3wJjwInCO7/L/50/nf2GvDu6P3u
    9+xt5QHh8Oa168Hu4/LH9YL7D/P326rp6e+I7koFufzS7cX0yQxxAK7qpPB98SrvkgikBUT97A+q
    8DHlxuCF2r7qQNpF2SbirNnp09DTNNGUzgXN6dCE0+3T39Ggz1LNbsv6x0/Jd8j4xCbE/8WByvbR
    ac8o0gLSQ9EWzf/QXNYm1nvXM8crxRnAvtb335DTBefF6ifsA+Nt4rrgC9mr3iDaCdo75ADlAOZ4
    5kvYGdcb2S7WGc7pzjfL5tYc4nPpXOQjzdjKfMVlw9vEmMTmzBzPhc54zPnQp9HJ1IjXnNVn0oXT
    odVpyjbJ69BhzWrDSL9nwOm/8MGbwPPBQ8JawuPCg8NuxBbFm8PuwhjJGcoyyTvEasBHxLfID8hH
    zBLU9NUl1pHVidYd35nhDOgh7LDtXfa699bxheHo4o/gbuAp52Xfp/iB/e/0C+mc5JLjcOXz/BXt
    Ct/e3Nbfuuq47UjquPcFHLQJZ+72/pr2S/Jz6SAZ0EayFl787eHP03DesdMW3VnmvuNL7XfnLej9
    5ZrjV+fj4SvgPdr73ILiPPEe/Xv3gfas8E/lyOpU7rEL4A4OE4MJRNcC2JXqQfKE6Cvwa/Bw8gLo
    NuU297rkYeMX3LHaENii1mPdsdXU15rQ0Mse0UvQNM6IzBvQPNWi0+HKeMiDzGfMPdj02ibUxtcj
    … truncated for brevity ….
    </swe:values
</swe:DataArray>
```

Finally, it is also possible to nest arrays, to define the encoding of the data as a standard MIME type, such as jpeg, and to reference an "out-of-band" block of data on the web or as an attachment within a SOAP message, as in this example:

```
<swe:DataArray definition="urn:ogc:def:property:OGC:scans">
    <swe:elementCount>
        <swe:Count definition="urn:ogc:def:property:OGC:numberOfScans">1080</swe:Count>
    </swe:elementCount>
    <swe:elementType name="scanLine>
        <swe:DataArray definition="urn:ogc:def:property:OGC:scanLine">
            <swe:elementCount>
                <swe:Count definition="urn:ogc:def:property:OGC:numberOfPixels">720</swe:Count>
            </swe:elementCount>
            <swe:elementType name="radiance">
                <swe:DataRecord>
                    <swe:field name="band1">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                    </swe:field>
                    <swe:field name="band2">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                    </swe:field>
                    <swe:field name="band3">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                    </swe:field>
                    <swe:field name="band4">
```

```
                        <swe:Quantity definition="urn:ogc:def:property:OGC:DN"/>
                    /swe:field>
                </swe:DataRecord>
            </swe:elementType>
            <swe:encoding>
                <swe:StandardFormat mimeType=”image/jpeg”/>
            </swe:encoding>
            <swe:values xlink:href=”http://www.mydomain.com/images/satPhoto1.jpg”/>
        </swe:DataArray>
</swe:DataArray>
```

9.2.5 Curves

The *Curve* element is derived from *AbstractArrayType* and is part of the *AnyData* group. It allows one to define one or more axes (or fields) for a curve (taking *AnyScalar* for each *field* value) and to provide a series of coordinate values using the *values* properties. An example of a calibration curve mapping temperature to resistance is given below.

```
<swe:Curve definition="urn:ogc:def:property:OGC:calibration">
    <swe:elementCount>
        <swe:Count definition="urn:ogc:def:property:OGC:numberOfSamples">21</swe:Count>
    </swe:elementCount>
    <swe:elementType>
        <swe:SimpleDataRecord>
            <swe:field name="temperature">
                <swe:Quantity definition="urn:ogc:def:property:OGC:temperature">
                        <swe:uom code="Cel"/>
                </swe:Quantity>
            </swe:field>
            <swe:field name="resistance">
                <swe:Quantity id="RES" definition="urn:ogc:def:property:OGC:resistance" >
                        <swe:uom code="Ohm"/>
                </swe:Quantity>
            </swe:field>
        </swe:SimpleDataRecord>
    </swe:elementType>
    <swe:encoding>
        <swe:TextBlock tokenSeparator="&#x20;" blockSeparator="," decimalSeparator="."/>
    </swe:encoding>
    <swe:values>
        -40,328.4 -35,237.7 -30,173.9 -25,128.5 -20,95.89 -15,72.23 -10,54.89 -5,42.07 0,32.51 5,25.31
        10,19.86 15,15.69 20,12.49 25,10 30,8.06 35,6.536 40,5.331 45,4.373 50,3.606 55,2.989 60,2.49
    </swe:values>
</swe:Curve>
```

In this example, the *definition* property defines two *Quantity* axes (i.e., temperature and resistance). The *values* property allows one to provide an array of the coordinate values using an efficient data block. Within the *values* block, are 21 pairs of temperature and resistance values, with white space separating tuples (i.e., the value pair in this case) and commas separating the tokens within a tuple. *Curve* can support axes of different types, for example, to map a *Count* to a *Category* or perhaps an array index (*Count*) to a *Quantity* value.

A *NormalizedCurve* is also defined in the SWE common parameters.xsd schema, which allows one to provide bias and gain values for either or both of the input and output axes of the curve.

9.2.6 TimeAggregates

This listing for a TimeGrid describes a 100 element temporal grid, indexed (3-102), with 12 hour offsets between points. The temporal position corresponding with index=0 is explicit.

```
<swe:TimeInstantGrid>
  <swe:extent>
    <swe:TimeGridEnvelope>
      <swe:low>3</swe:low>
      <swe:high>102</swe:high>
    </swe:TimeGridEnvelope>
  </swe:extent>
  <swe:originPos>2007-01-29T12:00:00.00+09:00</swe:originPos>
  <swe:offsetDuration>PT12H</swe:offsetDuration>
  <swe:duration>P50D</swe:duration>
</swe:TimeInstantGrid>
```

9.2.7 Phenomenon

This listing shows the application of the phenomenon schema to describe a library of standard phenomenon or observed properties. Example entries are provided for *Phenonmenon*, *ConstrainedPhenomenon*, and *CompositePhenonmenon*.

```
<?xml version="1.0" encoding="UTF-8"?>
<gml:Dictionary xmlns:gml="http://www.opengis.net/gml" xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/swe/1.0 ../swe.xsd" gml:id="phenomena_swe">
    <gml:description>
        A dictionary of phenomena, compiled through OWS-1, OWS-1.2 OWS-3.SJDC 2005-10-03
    </gml:description>
    <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">urn:ogc:def:property:OGC:ALL</gml:name>
    <gml:name>OWS Phenomena</gml:name>

    <!-- ======================================-->
    <!-- ===== Simple Phenomenon examples ======== -->
    <!-- ====================================== -->
    <gml:dictionaryEntry>
        <swe:Phenomenon gml:id="AtmosphericPressure">
            <gml:description xlink:href="http://sweet.jpl.nasa.gov/ontology/property.owl#AtmosphericPressure">
                fluid pressure exerted due to the gravitational effect on the column of atmosphere above the
                position of interest
            </gml:description>
            <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
                urn:ogc:def:property:OGC:AtmosphericPressure
            </gml:name>
            <gml:name>Atmospheric Pressure</gml:name>
        </swe:Phenomenon>
    </gml:dictionaryEntry>
    <!-- ============================-->
    <gml:dictionaryEntry>
        <swe:Phenomenon gml:id="Temperature">
            <gml:description xlink:href="http://sweet.jpl.nasa.gov/ontology/property.owl#Temperature"/>
            <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
                urn:ogc:def:property:OGC:Temperature
            </gml:name>
            <gml:name>Temperature</gml:name>
        </swe:Phenomenon>
    </gml:dictionaryEntry>

    <!-- ======================================-->
    <!-- ===== Constrained Phenomenon example ===== -->
```

```
<!-- ===================================== -->
<gml:dictionaryEntry>
    <swe:ConstrainedPhenomenon gml:id="WaterTemperature">
        <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
            urn:ogc:def:property:OGC:WaterTemperature
        </gml:name>
        <gml:name>Water Temperature</gml:name>
        <swe:base xlink:href="#Temperature"/>
        <swe:singleConstraint>
            <swe:TypedValue>
                <swe:property codeSpace="./">Medium</swe:property>
                <swe:value xsi:type="gml:CodeType" codeSpace="http://www.opengis.net/ows/material">
                    Water
                </swe:value>
            </swe:TypedValue>
        </swe:singleConstraint>
    </swe:ConstrainedPhenomenon>
</gml:dictionaryEntry>
<!-- =====================================-->
<gml:dictionaryEntry>
    <swe:ConstrainedPhenomenon gml:id="SurfaceWaterTemperature">
        <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
            urn:ogc:def:property:OGC:SurfaceWaterTemperature
        </gml:name>
        <gml:name>Surface Water Temperature</gml:name>
        <swe:base xlink:href="#WaterTemperature"/>
        <swe:singleConstraint>
            <swe:TypedValue>
                <swe:property codeSpace="./">Depth</swe:property>
                <swe:value>
                    <swe:Interval>
                        <swe:lowerBound xsi:type="gml:MeasureType" uom="./units.xml#m">
                            0.0
                        </swe:lowerBound>
                        <swe:upperBound xsi:type="gml:MeasureType" uom="./units.xml#m">
                            1.5
                        </swe:upperBound>
                    </swe:Interval>
                </swe:value>
            </swe:TypedValue>
        </swe:singleConstraint>
    </swe:ConstrainedPhenomenon>
</gml:dictionaryEntry>

<!-- =====================================-->
<!-- ===== Composite Phenomenon example ====== -->
<!-- ===================================== -->
<gml:dictionaryEntry>
    <gml:name>Earthquake Parameters</gml:name>
        <swe:component>
            <swe:CompositePhenomenon gml:id="EarthquakeLocation" dimension="3">
                <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
                    urn:ogc:def:property:OGC:EarthquakeLocation
                </gml:name>
                <gml:name>Earthquake Location</gml:name>
                <swe:component xlink:href="#Epicentre"/>
                <swe:component xlink:href="#Depth"/>
                <swe:component xlink:href="#OriginTime"/>
            </swe:CompositePhenomenon>
        </swe:component>
        <swe:component>
            <swe:CompositePhenomenon gml:id="MomentTensor" dimension="6">
                <gml:name codeSpace="urn:ogc:tc:arch:doc-rp(05-010)">
                    urn:ogc:def:property:OGC:MomentTensor
                </gml:name>
                <gml:name>Earthquake Moment Tensor</gml:name>
                <swe:component xlink:href="#Mrr"/>
```

```
                    <swe:component xlink:href="#Mtt"/>
                    <swe:component xlink:href="#Mff"/>
                    <swe:component xlink:href="#Mrt"/>
                    <swe:component xlink:href="#Mrf"/>
                    <swe:component xlink:href="#Mtf"/>
                </swe:CompositePhenomenon>
            </swe:component>
        </swe:CompositePhenomenon>
    </gml:dictionaryEntry>
```

10 SensorML XML Encodings and Examples (Informative)

The primary focus of SensorML is to define processes and processing components associated with the measurement and post-measurement transformation of observations. All processes share a common set of base elements, whether the process is atomic or composite, physical or non-physical.

ProcessType elements in SensorML are derived from the *gml:AbstractFeatureType* and thus inherit *gml:description* and *gml:name* properties, as well as the *gml:id* attribute. In addition, all processes include *input*, *output*, and *parameter* properties, as well as a rich set of metadata. The *input*, *output*, and *parameter* properties take *swe:anyData* for their values, providing consistency between process descriptions and descriptions of observations and services throughout the SWE framework.

As discussed in the model section, processes can be classified as atomic processes, which are considered to be indivisible, and composite processes, which consist of a collection of processes that can be linked together for the purpose of providing desired output. In addition, processes can be classified as physical processes, if the location and physical interface of these processes are relevant, and non-physical processes, if location and physical interface are irrelevant.

10.1 ProcessModel (Atomic Non-Physical Process)

The main purpose of the *ProcessModel* is to provide an atomic, potentially executable process description of processes where physical location and physical interfaces for the process itself are not important. While *ProcessModel* provides metadata that is useful for discovery and assistance to humans, the properties that are critical for supporting execution of the *ProcessModel* within SensorML-enabled software are the *inputs*, *outputs*, *parameters*, and *method* properties.

The following example describes the process of calculating wind chill factor from temperature and wind speed. When associated with software implementation or MathML algorithm description, defined within the *method* property, it serializes an executable process for deriving higher-level products from low-level sensor observations. This example has minimal metadata presented and no *parameters* required.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sensorML/1.0
    http://schemas.opengis.net/sensorML/1.0.0/sensorML.xsd" version="1.0">
    <sml:member xlink:arcrole="urn:ogc:def:role:process">
        <sml:ProcessModel gml:id="WINDCHILL_PROCESS">
            <!-- METADATA SECTION -->
            <gml:description>Wind chill temperature computation process</gml:description>
            <!-- INPUTS DEFINITION -->
            <sml:inputs>
                <sml:InputList>
                    <sml:input name="atmosphericConditions">
```

```
                        <swe:DataRecord>
                            <swe:field name="ambient_temperature">
                                <swe:Quantity definition="urn:ogc:def:property:OGC:temperature">
                                    <swe:uom code="degF"/>
                                </swe:Quantity>
                            </swe:field>
                            <swe:field name="wind_speed">
                                <swe:Quantity definition="urn:ogc:def:property:OGC:windSpeed">
                                    <swe:uom code="mph"/>
                                </swe:Quantity>
                            </swe:field>
                        </swe:DataRecord>
                    </sml:input>
                </sml:InputList>
            </sml:inputs>
            <!-- OUTPUTS DEFINITION -->
            <sml:outputs>
                <sml:OutputList>
                    <sml:output name="windchill_temperature">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:temperature">
                            <swe:uom code="degF"/>
                        </swe:Quantity>
                    </sml:output>
                </sml:OutputList>
            </sml:outputs>
            <!-- METHOD DEFINITION -->
            <sml:method xlink:href="urn:ogc:def:process:WindChill:1.0"/>
        </sml:ProcessModel>
    </sml:member>
</sml:SensorML>
```

10.1.1 ProcessMethod

The *method* property within a *ProcessModel* (and a *Component*, as well) provides several important pieces of information for validating and possibly executing individual (atomic) processes. The method property takes a *ProcessMethod* object that provides rules for validating instances of the process, an algorithm to be used for execution of this process model, and one or more reference implantations of software for on-demand execution of the process.

The algorithm can be defined using the content version (the most rigorous) of MathML, so that the software is able to generate a new model dynamically. If no MathML is provided, the software can rely on a local implementation of this method provided by the *implementation* property of *ProcessMethod*, or download a particular implementation adapted to the particular platform and processing framework used. For this last purpose, the *ProcessMethod* object can also contain a list of approved implementations, which can include source and/or binary code for a specific combination of language, platform and processing framework.

Each *ProcessMethod* is also identified by a unique URI (typically a URN), so that software can relate it to a previously cached method description and/or implementation. A *ProcessModel* will thus typically use this URI to point to a method defined elsewhere, rather than include the whole definition inline. The software can then decide to download the whole method definition again, or simply rely on the URI to find a local matching implementation. It is expected that a *ProcessMethod* will be defined and discoverable within online registries or other online resources.

Finally, in order to make sure that the *ProcessModel* inputs, outputs and parameters are defined according to what is specified in the method, the *ProcessMethod rules* property encapsulates a rule set defined by some rule-based language, such as XML RelaxNG or Schematron. This finer grain rule set can be used in addition to the SensorML XML Schema to further validate a specific *ProcessModel* as well as to automatically generate new instances of specific processes using the chosen method. Such a rules set should rigorously define name, definition and unit of measure for each input, output and parameter, as well as any metadata appropriate for that process.

An example of a *ProcessMethod* instance for the previously provided wind chill *ProcessModel* is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml: ProcessMethod xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sensorML/1.0
    http://schemas.opengis.net/sensorML/1.0.0/process.xsd" version="1.0">
    <!-- Description -->
    <gml:description>Method defining the Wind Chill Temperature calculation process</gml:description>
    <!-- Contact -->
    <sml:contact xlink:arcrole="author">
        <sml:ResponsibleParty>
            <sml:individualName>Alexandre Robin</sml:individualName>
            <sml:organizationName>University of Alabama in Huntsville</sml:organizationName>
            <sml:positionName>Research Scientist</sml:positionName>
            <sml:contactInfo>
                <sml:phone>
                    <sml:voice>(256) 961-7978</sml:voice>
                </sml:phone>
                <sml:address>
                    <sml:electronicMailAddress>robin@nsstc.uah.edu</sml:electronicMailAddress>
                </sml:address>
            </sml:contactInfo>
        </sml:ResponsibleParty>
    </sml:contact>
    <!-- Documentation -->
    <sml:documentation xlink:arcrole="formula">
        <sml:Document>
            <gml:description>NWS document giving wind chill temperature formula</gml:description>
            <sml:onlineResource xlink:href="http://www.wrh.noaa.gov/slc/projects/wxcalc/windChill.pdf"/>
        </sml:Document>
    </sml:documentation>
    <!-- Rules Set -->
    <sml:rules>
        <sml:RulesDefinition>
            <gml:description>1 input must be "ambiant_temperature" in degF. 1 input must be "wind_speed" in
                mph. 1 output must be "windchill_temperature" in degF. No parameters
            </gml:description>
            <sml:relaxNG xlink:href="http://vast.uah.edu/SensorML/profile/process/WindChill.rng"/>
        </sml:RulesDefinition>
    </sml:rules>
    <!-- Algorithm -->
    <sml:algorithm>
        <sml:AlgorithmDefinition>
            <gml:description>Apply following formula: Tc = 35.74 + 0.6215*T - 35.75*V^0.16 + 0.4275*T*V^0.16
            </gml:description>
            <sml:mathML xlink:href="http://vast.uah.edu/SensorML/process/math/WindChill_Algorithm.xml"/>
        </sml:AlgorithmDefinition>
```

```
    </sml:algorithm>
    <!-- Implementations -->
    <sml:implementation>
        <sml:ImplementationCode language="java" framework="uah-dpf" version="1.0">
            <sml:sourceRef xlink:href="http://vast.uah.edu/SensorML/process/src/WindChill_Process.java"/>
            <sml:binaryRef xlink:href="http://vast.uah.edu/SensorML/process/bin/WindChill_Process.class"/>
        </sml:ImplementationCode>
    </sml:implementation>
</sml:ProcessMethod>
```

A description and example of RelaxNG and MathML is beyond the scope of this document.

10.2 Component (Atomic Physical Process)

A *Component* describes a physical process that is, by desire or necessity, considered to be indivisible (i.e. it is atomic). Example Component types include, for example, all transducers (i.e. detectors and actuators), clocks, batteries, and filters. In addition to the properties of the *ProcessModel* described above, a *Component* can include properties describing the location and physical interfaces of the physical process.

Specifically, the additional properties include *spatialReferenceFrame* which provides a local Coordinate Reference System for the component using *gml:EngineeringCRS*. It is the state (e.g. location, orientation, and velocity) of this local spatial frame relative to some external spatial reference frame that is provided by the *location* or *position* properties of the *Component*. The *gml:EngineeringCRS* provides reference IDs and definitions of the frame axes, as well as a textual description of the origin and orientation of the local spatial frame. This in turn provides system integrators with an explicit understanding of the spatial reference frames to use when providing component mounting angles and distances.

For specifying the spatial position, a *Component* can take either a *location* property which can take a *gml:Point* or *gml:_Curve* as its value, or a *position*, which can take either a *swe:Position*, *swe:Vector*, or *sml:_Process* as its value. The *position* property is considered more robust since it can provide a more complete dynamic state of the object, including time-tagged location, orientation, velocity, acceleration, angular velocity, and angular acceleration. In addition, one can define the location of a *Component* using a *Process* or another *Component* (e.g. a GPS).

Additionally, the *temporalReferenceFrame* property takes a *gml:TemporalCRS* as its value. It is used to specify a local time frame that might be used internally by the process. This might include, for example, scan-start time within a image scanner, or the time frame provided by an internal clock. Such local time reference systems can be related to external time frames through the *timePosition* property, which takes either a *swe:Time* or *sml:_Process* as its value..

Finally, any physical interfaces to the component can be described using the interfaces property. The *InterfaceDefinition* mimics the OSI interface stack, with any of the OSI layers being optional.

The following example is for a Davis 7817 Thermometer used in a weather station. The *inputs* and *outputs* are simple, each taking a single *ObservableProperty* as its value; real temperature for input and electrical resistance for output. There are three parameters for the thermometer, including a steady-state (i.e. calibration) curve, an error curve, and a latency time response in air. The *method* for this *Component* is a general detector method that has been defined elsewhere and is referenced here using the *xlink:href* association.

The metadata section of this example has been omitted for brevity and is provided and discussed in Section 11.5.

```
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sensorML/1.0
    http://schemas.opengis.net/sensorML/1.0.0/sensorML.xsd" version="1.0">
    <member xlink:role="urn:ogc:def:role:OGC:detector">
        <Component gml:id="Davis_7817 ">
            <gml:name>Davis 7817 Thermometer</gml:name>
            <!--                                                    -->
            <!-- metadata omitted for brevity, see Section 11.5 for associated metadata example  -->
            <!--                                                    -->
            <!--~~~~~~~~~~~~~~~~~~~~~~~~~~-->
            <!--Detector Coordinate Frame-->
            <!--~~~~~~~~~~~~~~~~~~~~~~~~~~-->
            <spatialReferenceFrame>
                <gml:EngineeringCRS gml:id="THERMOMETER_FRAME">
                    <gml:srsName>Thermometer Detector Frame</gml:srsName>
                    <gml:usesCS xlink:href="urn:ogc:def:crs:ogc:1.0:xyzFrame"/>
                    <gml:usesEngineeringDatum>
                        <gml:EngineeringDatum gml:id="Davis_7817_Datum">
                            <gml:datumName>Sensor Datum</gml:datumName>
                            <gml:anchorPoint>
                                origin is at the tip of the thermometer;
                                x and Y are orthogonal to z but undetermined;
                                z is along the long axis of symmetry of the thermometer
                            </gml:anchorPoint>
                        </gml:EngineeringDatum>
                    </gml:usesEngineeringDatum>
                </gml:EngineeringCRS>
            </spatialReferenceFrame>
            <!--~~~~~~~~~~~~~~~~~~~~~~~~~~-->
            <!-- Interface Definition              -->
            <!--~~~~~~~~~~~~~~~~~~~~~~~~~~-->
            <interfaces>
                <InterfaceList>
                    <interface name="cable">
                        <InterfaceDefinition>
                            <physicalLayer>
                                <swe:DataRecord definition="urn:ogc:def:property:OGC:interface">
                                    <swe:field name="cableLength">
                                        <swe:Quantity definition="urn:ogc:def:property:OGC:length">
                                            <swe:uom code="mm"/>
                                            <swe:value>7.6</swe:value>
                                        </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="connectorType">
                                        <swe:Category
                                            definition="urn:ogc:def:property:OGC:electricalConnector">
                                            <swe:value>RJ-11</swe:value>
                                        </swe:Category>
```

```
                        </swe:field>
                        <swe:field name="cableType">
                            <swe:Category definition="urn:ogc:def:property:OGC:electricalCable">
                                <swe:value>26-AWG:4-conductor</swe:value>
                            </swe:Category>
                        </swe:field>
                    </swe:DataRecord>
                </physicalLayer>
            </InterfaceDefinition>
        </interface>
    </InterfaceList>
</interfaces>
<!--~~~~~~~~~~~~~~~-->
<!-- Detector Inputs-->
<!--~~~~~~~~~~~~~~~-->
<!-- note: in most cases, a detector's input will be a phenomenon and its output a measured
    digital value (e.g. Quantity)  -->
<inputs>
    <InputList>
        <input name="temperature">
            <swe:ObservableProperty definition="urn:ogc:def:property:OGC:temperature"/>
        </input>
    </InputList>
</inputs>
<!--~~~~~~~~~~~~~~~~-->
<!-- Detector Outputs-->
<!--~~~~~~~~~~~~~~~~-->
<!-- note: in this case, the detector's output is also a phenomenon (electrical resistance) rather than
    measured digital values -->
<outputs>
    <OutputList>
        <output name="measuredTemperature">
            <swe:ObservableProperty definition="urn:ogc:def:property:OGC:resistance"/>
        </output>
    </OutputList>
</outputs>
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!--  Temperature Response     -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<parameters>
    <ParameterList>
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
        <!--  Calibration (Steady-State) Curve    -->
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
        <parameter name="steadyState" xlink:arcrole="urn:ogc::def:property:OGC:calibration">
            <swe:NormalizedCurve fixed="true" definition="urn:ogc:def:property:OGC:steadyState">
                <swe:function>
                    <swe:Curve>
                        <swe:elementCount>
                            <swe:Count>
                                <swe:value>21</swe:value>
                            </swe:Count>
                        </swe:elementCount>
                        <swe:elementType>
                            <swe:SimpleDataRecord>
                                <swe:field name="temperature">
                                    <swe:Quantity gml:id="TEMP1"
                                        definition="urn:ogc:def:property:OGC:temperature">
                                        <swe:uom code="cel"/>
                                    </swe:Quantity>
                                </swe:field>
                                <swe:field name="resistance">
                                    <swe:Quantity gml:id="RES"
                                        definition="urn:ogc:def:property:OGC:resistance">
                                        <swe:uom code="kohm"/>
                                    </swe:Quantity>
                                </swe:field>
```

```xml
                    </swe:SimpleDataRecord>
                </swe:elementType>
                <swe:values>
                    -40,328.4 -35,237.7 -30,173.9 -25,128.5 -20,95.89 -15,72.23
                    -10,54.89 -5,42.07 0,32.51 5,25.31 10,19.86 15,15.69
                    20,12.49 25,10 30,8.06 35,6.536 40,5.331 45,4.373 50,3.606
                    55,2.989 60,2.49
                </swe:values>
            </swe:Curve>
        </swe:function>
    </swe:NormalizedCurve>
</parameter>
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!--  Accuracy Curve    -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<parameter name="accuracy" xlink:arcrole="urn:ogc:def:property:OGC:accuracy">
    <swe:NormalizedCurve fixed="true" definition="urn:ogc:def:property:OGC:absoluteError">
        <swe:function>
            <swe:Curve>
                <swe:elementCount>
                    <swe:Count>
                        <swe:value>6</swe:value>
                    </swe:Count>
                </swe:elementCount>
                <swe:elementType>
                    <swe:SimpleDataRecord>
                        <swe:field name="temperature">
                            <swe:Quantity gml:id="TEMP2"
                                definition="urn:ogc:def:property:OGC:temperature">
                                <swe:uom code="cel"/>
                            </swe:Quantity>
                        </swe:field>
                        <swe:field name="absoluteError">
                            <swe:Quantity gml:id="ERR"
                                definition="urn:ogc:def:property:OGC:absoluteError">
                                <swe:uom code="cel"/>
                            </swe:Quantity>
                        </swe:field>
                    </swe:SimpleDataRecord>
                </swe:elementType>
                <swe:values>
                    -40,0.0 -20,0.1 -10,0.2 0,0.3 10,0.4 20,0.5
                </swe:values>
            </swe:Curve>
        </swe:function>
    </swe:NormalizedCurve>
</parameter>
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!--  Latency in Air Value                -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<parameter name="latency" xlink:arcrole="urn:ogc:def:property:OGC:latencyTime">
    <ConditionalValue>
        <condition name="medium">
            <swe:Category definition="urn:xogc:def:property:OGC:medium">
                <swe:value>air</swe:value>
            </swe:Category>
        </condition>
        <data>
            <swe:Quantity definition="urn:ogc:def:property:OGC:duration">
                <swe:uom code="s"/>
                <swe:value>10</swe:value>
            </swe:Quantity>
        </data>
    </ConditionalValue>
</parameter>
</ParameterList>
</parameters>
```

```
            <method xlink:href="urn:ogc:def:process:OGC:detector"/>
        </Component>
    </member>
</SensorML>
```

10.3 ProcessChain (Composite Non-Physical Process)

A *ProcessChain* is a collection of linked processes that in and of itself, does not have a connection to the physical realm. In other words, while a *ProcessChain* may have physical processes as components, the location of the *ProcessChain*, itself, is considered irrelevant.

Whereas *ProcessModel* and Component define an atomic processes that can be executed using algorithms, code, or physical operations defined by the *method* property, a *ProcessChain* defines a process chain where the chain itself defines the execution methodology. *ProcessChain* is itself a SensorML process and thus includes the elements of *metadataGroup*, *inputs*, *outputs*, and *parameters*. In addition, a *ProcessChain* includes the *components* property which defines or links to processes used within the chain, as well as a *connections* property that defines the connections between the internal processes.

The steps for defining a *ProcessChain* are:

- Define the metadata, *inputs*, *outputs*, and *parameters* for the process chain itself. This exposes those data components that are accessible and linkable by components external to this process chain
- Define, or reference through *xlink:href*, all of the *processes* that are used inside of this *ProcessChain*
- Define all *connections* between process *inputs*, *outputs*, and *parameters* throughout the chain, including the *inputs*, *outputs*, and *parameters* of the *ProcessChain* itself

The following example defines a simple process chain for determining the minimum and maximum values from a stream of values. It outputs a new maximum value whenever the current maximum threshold is exceeded by the input value, and outputs a new minimum value whenever a value lower than the current minimum threshold is detected. Such a process could describe for example, the process by which the daily minimum and maximum temperatures are determined within a weather station console.

While such a process could be defined using an atomic *ProcessModel*, this example illustrates a simple process chain which utilizes two instances of the same generic *ProcessModel*; that is, a *ProcessModel* that compares a single value to a threshold value. It also illustrates that the *output* of a process can alter the *parameter* values of any process.

The first part of this example defines the metadata (omitted), *inputs*, *outputs*, and *parameters* of the *ProcessChain* itself. Only those *inputs*, *outputs*, and *parameters* defined at the *ProcessChain* level are scoped to be accessible to outside processes or applications.

```
<?xml version="1.0"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sensorML/1.0
    http://schemas.opengis.net/sensorML/1.0.0/sensorML.xsd" version="1.0">
    <member xlink:role="urn:ogc:def:role:OGC:processChain">
        <ProcessChain gml:id="MinMax">
            <gml:description>
                Input accepts a continuous stream of values;
                Outputs to the "newMaximum" channel any value that
                  exceeds the current maximum value;
                Outputs to the "newMinimum" channel any value that
                  is less than the current minimum value;
                Retains the current minimum and maximum values until
                  the value of "true" is received on the "reset" parameter;
            </gml:description>
            <gml:name>Minimum-Maximum Detector</gml:name>
            <!--~~~~~~~~~~~~~~~~~~~~~~-->
            <!--Process Chain Inputs      -->
            <!--~~~~~~~~~~~~~~~~~~~~~~-->
            <inputs>
                <InputList>
                    <input name="inputValue">
                        <swe:Quantity/>
                    </input>
                </InputList>
            </inputs>
            <!--~~~~~~~~~~~~~~~~~~~~~~~~-->
            <!--Process Chain Outputs     -->
            <!--~~~~~~~~~~~~~~~~~~~~~~~~-->
            <outputs>
                <OutputList>
                    <output name="newMinimum">
                        <swe:Quantity/>
                    </output>
                    <output name="newMaximum">
                        <swe:Quantity/>
                    </output>
                </OutputList>
            </outputs>
            <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
            <!--Process Chain Parameters        -->
            <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
            <parameters>
                <ParameterList>
                    <parameter name="reset">
                        <swe:Boolean/>
                    </parameter>
                </ParameterList>
            </parameters>
```

The second part of the example defines the processes that are a component of this process chain. In this case, we define two instances of the same ProcessModel, one for determining values greater than a threshold (“maxCompare”) and one for detecting values less than a threshold (“minCompare”). For these instances, the logic is defined and in this case “fixed” using the “logic” parameter.

```
<components>
    <ComponentList>
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
        <!--  Maximum Compare Process        -->
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
        <component name="maxCompare" xlink:arcrole="urn:ogc:def:role:process">
            <ProcessModel>
                <inputs>
                    <InputList>
                        <input name="inputValue">
                            <swe:Quantity/>
                        </input>
                    </InputList>
                </inputs>
                <outputs>
                    <OutputList>
                        <output name="newValue">
                            <swe:Quantity/>
                        </output>
                    </OutputList>
                </outputs>
                <parameters>
                    <ParameterList>
                        <parameter name="threshold">
                            <swe:Quantity/>
                        </parameter>
                        <parameter name="logic">
                            <swe:Category definition="urn:ogc:def:property:OGC:logic">
                                <swe:value>GreaterThan</swe:value>
                            </swe:Category>
                        </parameter>
                        <parameter name="reset">
                            <swe:Boolean/>
                        </parameter>
                    </ParameterList>
                </parameters>
                <method xlink:href="urn:ogc:def:property:OGC:logicalCompare001"/>
            </ProcessModel>
        </component>
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
        <!--  Minimum Compare Process           -->
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
        <component name="minCompare" xlink:arcrole="urn:ogc:def:role:process">
            <ProcessModel>
                <inputs>
                    <InputList>
                        <input name="inputValue">
                            <swe:Quantity/>
                        </input>
                    </InputList>
                </inputs>
                <outputs>
                    <OutputList>
                        <output name="newValue">
                            <swe:Quantity/>
                        </output>
                    </OutputList>
                </outputs>
                <parameters>
                    <ParameterList>
                        <parameter name="threshold">
                            <swe:Quantity/>
                        </parameter>
                        <parameter name="logic">
                            <swe:Category definition="urn:ogc:def:property:OGC:logic">
                                <swe:value>GreaterThan</swe:value>
                            </swe:Category>
```

```
                    </parameter>
                    <parameter name="reset">
                        <swe:Boolean/>
                    </parameter>
                </ParameterList>
            </parameters>
            <method xlink:href="urn:ogc:def:process:OGC:logicalCompare001"/>
        </ProcessModel>
    </component>
</ComponentList>
</components>
```

The last part of the *ProcessChain* example defines the *connections* between *inputs*, *outputs*, and *parameters*. The *connection* property uses a *Link* object to reference the *source* and *destination* of a connector.

To reference a particular data component inside of a *ProcessChain*, SensorML defines a syntax using property *qnames* as path components starting at the base of the *ProcessChain* instance. Thus all path references will begin with the process name, followed by either *inputs*, *outputs*, *parameters*, or *components*, indicating those properties within that process. The name "this" is reserved to indicate the process that encloses the connections. A path only uses names of properties (lowerCamelCase elements) and not Object names. If the property has a value assigned to the *name* attribute, then that will be used in place of the element name, otherwise the path will use the element name.

Issue Name: Use of XPath for Connection Links. (meb, 2007-02-22)]

Issue Description: The use of XPath for defining links between Processes and DataSources has been considered and was rejected for the following reasons:

(1) unnecessarily complex,
(2) challenges with xlink:href associations, and
(3) links imply flow of data between processes and do not imply connections between xml elements (as is the case with XPath)

Resolution: a more abstract, but simple syntax devised that can be unambiguously parsed and interpreted

For example, a reference to the value of the first input component of the "minMax" *ProcessChain* would be "*this/inputs/inputValue*" while the reference to the input value of the "maxCompare" process would be "*maxCompare/inputs/inputValue*".

```
<connections>
    <ConnectionList>
        <!-- process "value" input to maxCompare "inputValue" input -->
        <connection name="inputToMaxInput">
            <Link>
                <source ref="this/inputs/inputValue"/>
                <destination ref="maxCompare/inputs/inputValue"/>
            </Link>
        </connection>
        <!-- process "value" input to minCompare "inputValue" input -->
```

```
                    <connection name="inputToMinInput">
                        <Link>
                            <source ref="this/inputs/inputValue"/>
                            <destination ref="minCompare/inputs/inputValue"/>
                        </Link>
                    </connection>
                    <!-- maxCompare "newValue" output to process "newMaximum" output -->
                    <connection name="maxResultToProcessOut">
                        <Link>
                            <source ref="maxCompare/outputs/newValue"/>
                            <destination ref="process/outputs/newMaximum"/>
                        </Link>
                    </connection>
                    <!-- minCompare "newValue" output to process "newMinimum" output -->
                    <connection name="maxResultToProcessOut">
                        <Link>
                            <source ref="minCompare/outputs/newValue"/>
                            <destination ref="process/outputs/newMinimum"/>
                        </Link>
                    </connection>
                    <!-- maxCompare "newValue" output to maxCompare "threshold" parameter -->
                    <connection name="maxResultToMaxThreshold">
                        <Link>
                            <source ref="maxCompare/outputs/newValue"/>
                            <destination ref="maxCompare/parameters/threshold"/>
                        </Link>
                    </connection>
                    <!-- minCompare "newValue" output to minCompare "threshold" parameter -->
                    <connection name="minResultToMinThreshold">
                        <Link>
                            <source ref="minCompare/outputs/newValue"/>
                            <destination ref="minCompare/parameters/threshold"/>
                        </Link>
                    </connection>
                    <!-- process "reset" parameter to maxCompare "reset" parameter -->
                    <connection name="processResetToMaxReset">
                        <Link>
                            <source ref="this/parameters/reset"/>
                            <destination ref="maxCompare/parameters/reset"/>
                        </Link>
                    </connection>
                    <!-- process "reset" parameter to minCompare "reset" parameter -->
                    <connection name="processResetToMinReset">
                        <Link>
                            <source ref="this/parameters/reset"/>
                            <destination ref="minCompare/parameters/reset"/>
                        </Link>
                    </connection>
                </ConnectionList>
            </connections>
        </ProcessChain>
        <!-- ~~~~~~~~~~ End of Process Chain ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    </member>
</SensorML>
```

10.4 System (Composite Physical Process)

Like a *ProcessChain*, a *System* is a collection of processes that collectively result in a desired output of data. Unlike a *ProcessChain*, a *System* has a physical presence such that information about its spatial position and physical interface may be relevant to its application.

Thus a *System* includes positional information (spatial and temporal), allowing one to relate it and its components to the real world. It inherits all the properties of the *ProcessChain*, but like *Component*, it includes the additional properties for *spatialReferenceFrame*, *temporalReferenceFrame*, *position*, *location*, and *interface*, which were discussed previously. Additionally, System has a *positions* property which provides a means to relate the relative position of its components to one another or to some external reference system.

The following example is for a particular instance of a Davis weather station. The station consist of multiple detectors including a thermometer (provided in Section 11.2), an anemometer, a barometer, a wind gauge, wind vane, and system clock. In addition, the weather station includes a process (provided in Section 11.3) that determines and outputs the daily minimum and maximum temperatures as the day progresses. In this example, the various components are not defined inline, but are instead referenced through an *xlink:href* association. As with previous examples, metadata for the station has been omitted for the sake of brevity and the instance is broken into relevant pieces interspersed with discussion.

As with all processes the *System* itself can include a *gml:description* and *gml:name*, as well as a full set of metadata.

```
<?xml version="1.0"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sensorML/1.0
    http://schemas.opengis.net/sensorML/1.0.0/sensorML.xsd" version="1.0">

    <member xlink:role="urn:ogc:def:role:OGC:sensorSystem">
        <System gml:id="Davis_Simple_Station">
            <gml:description>Davis Weather Station located at NSSTC on UAH Campus</gml:description>
            <gml:name>Weather Station UAH:0024</gml:name>
            <!--                                                    -->
            <!-- metadata omitted for brevity, see Section 11.5 for associated metadata example  -->
            <!--
                                                -->
```

A System description includes definition of its reference frame. As with Component, the position of the System defines the relationship of its reference frame to some external reference system (e.g. the position relative to a geospatial CRS or relative to the reference frame of some other System). In addition, the System's reference frame provides a means by which components of the system can be located and oriented relative to the station's position.

The station's location and orientation can be specified using the *position* or *location* property, or they can be specified along with the component's positions within the *PositionList* that is part of the *positions* property. This example uses the second method.

```
<!--~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- System Coordinate Frame -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<spatialReferenceFrame>
    <gml:EngineeringCRS gml:id="STATION_FRAME">
        <gml:srsName>Station Frame</gml:srsName>
        <gml:usesCS xlink:href="urn:ogc:cs:OGC:xyzFrame"/>
        <gml:usesEngineeringDatum>
            <gml:EngineeringDatum gml:id="STATION_DATUM">
                <gml:datumName>Station Datum</gml:datumName>
                <gml:anchorPoint>origin is at the base of the mounting;
                    x and y are orthogonal to z but undetermined;
                    z is along the axis of the mounting pole - typically vertical
                </gml:anchorPoint>
            </gml:EngineeringDatum>
        </gml:usesEngineeringDatum>
    </gml:EngineeringCRS>
</spatialReferenceFrame>
```

The physical and software interfaces of a *System* can be specified using the *interfaces* property. The *interface* definition can provide specific information regarding the physical flow and encoding of any data that has been described within the *System inputs*, *outputs*, and *parameters* properties. In this particular example, the interface is merely described as a RS232 port with a DB9 connector, without specifying exactly how the individual data components are associated with that interface.

```
<!--~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- System Interface          -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<interfaces>
    <InterfaceList>
        <interface name="RS-232">
            <InterfaceDefinition>
                <!-- http://www.interfacebus.com/Design_Connector_RS232.html -->
                <applicationLayer>
                    <swe:Category definition="urn:ogc:def:property:OGC:applicationLink">
                        <swe:value>urn:davis:def:protocol:weatherLink</swe:value>
                    </swe:Category>
                </applicationLayer>
                <physicalLayer>
                    <swe:DataRecord definition="urn:ogc:def:property:OGC:RS232">
                        <swe:field name="num-bits">
                            <swe:Count definition="urn:ogc:def:property:OGC:numberOfBits">
                                <swe:value>8</swe:value>
                            </swe:Count>
                        </swe:field>
                        <swe:field name="parity">
                            <swe:Boolean definition="urn:ogc:def:property:OGC:parity">
                                <swe:value>false</swe:value>
                            </swe:Boolean>
                        </swe:field>
                    </swe:DataRecord>
                </physicalLayer>
                <mechanicalLayer>
                    <swe:DataRecord definition="urn:ogc:def:property:OGC:DB9">
                        <swe:field name="pin-out">
                            <swe:Category definition="urn:ogc:def: property:OGC:pinout">
                                <swe:value>EIA574</swe:value>
                            </swe:Category>
                        </swe:field>
                    </swe:DataRecord>
                </mechanicalLayer>
```

```
                </InterfaceDefinition>
            </interface>
        </InterfaceList>
    </interfaces>
```

The *System* itself can have *inputs*, *outputs*, and *parameters*, as can all the component processes internal to the *System*. However, only those *inputs*, *outputs*, and *parameters* that are defined at the *System* level are scoped to be accessible to external processes or applications.

Those *parameters* existing at the *System* level, could be interpreted as being taskable or configurable properties of the system. In this example, there are no *parameters*. *Inputs* or *outputs* that are not of type *ObservableProperty* define the data that can flow into and out of the system. In this example, all inputs are of the type *ObservableProperty*. The *outputs* consist of two blocks of data, one for atmospheric state properties and the other for daily minimum and maximum temperatures.

```
<!--~~~~~~~~~~~~~~~-->
<!--System Inputs-->
<!--~~~~~~~~~~~~~~~-->
<inputs>
    <InputList>
        <input name="atmosphericTemperature">
            <swe:ObservableProperty definition="urn:ogc:def:property:OGC:temperature"/>
        </input>
        <input name="wind">
            <swe:ObservableProperty definition="urn:ogcdef:property:OGC:windSpeed"/>
        </input>
        <input name="rainfall">
            <swe:ObservableProperty definition="urn:ogc:def:property:OGC:rainfall"/>
        </input>
        <input name="atmosphericPressure">
            <swe:ObservableProperty definition="urn:ogcdef:property:OGC:pressure"/>
        </input>
    </InputList>
</inputs>
<!--~~~~~~~~~~~~~~~-->
<!--System Outputs-->
<!--~~~~~~~~~~~~~~~-->
<outputs>
    <OutputList>
        <output name="weatherMeasurements">
            <swe:DataRecord gml:id="outputGroup">
                <swe:field name="time">
                    <swe:Time definition="urn:ogc:def:property:OGC:observationTime">
                        <swe:uom xlink:href="urn:ogc:def:unit:ISO:8601"/>
                    </swe:Time>
                </swe:field>
                <swe:field name="temperature">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:temperature">
                        <swe:uom code="cel"/>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="windSpeed">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:windSpeed">
                        <swe:uom code="m/s"/>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="windDirection">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:windDirection">
                        <swe:uom code="deg"/>
```

```
                    </swe:Quantity>
                </swe:field>
                <swe:field name="rainfall">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:rainfall">
                        <swe:uom code="mm"/>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="barometricPressure">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:barometricPressure">
                        <swe:uom code="hPa"/>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="windChill">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:windChill">
                        <swe:uom code="cel"/>
                    </swe:Quantity>
                </swe:field>
            </swe:DataRecord>
        </output>
        <output name="minMax">
            <swe:DataRecord gml:id="minMaxTemperatures">
                <swe:field name="dailyMinTemp">
                    <swe:Quantity
                        definition="urn:ogc:def:property:OGC:dailyMinimumTemperature"/>
                </swe:field>
                <swe:field name="dailyMaxTemp">
                    <swe:Quantity
                        definition="urn:ogc:def:property:OGC:dailyMinimumTemperature"/>
                </swe:field>
            </swe:DataRecord>
        </output>
    </OutputList>
</outputs>
```

A *System* consists of one or more component processes or data sources. These are listed and perhaps defined within the *components* property. These individual *components* can either be defined inline within the *System* document or they can be referenced using *xlink:href* associations, as has been done in this example. Either way, the component process definitions should completely define appropriate *inputs*, *outputs*, and *parameters* which can then be linked within the *connections* property to follow.

```
<components>
    <ComponentList>
        <!--~~~~~~~~~~~~~~~~~~-->
        <!--  Clock   -->
        <!--~~~~~~~~~~~~~~~~~~-->
        <component name="clock" xlink:arcrole="urn:ogc:def:process:OGC:detector"
            xlink:href="urn:ogc:sensor:Davis:clock"/>
        <!--~~~~~~~~~~~~~~~~~~-->
        <!--  Barometric   -->
        <!--~~~~~~~~~~~~~~~~~~-->
        <component name="barometer" xlink:arcrole ="urn:ogc:def:process:OGC:detector"
            xlink:href="urn:ogc:sensor:Davis:7441"/>
        <!--~~~~~~~~~~~~~~~~~~-->
        <!--  Thermometer  -->
        <!--~~~~~~~~~~~~~~~~~~-->
        <component name="thermometer" xlink:arcrole="urn:ogc:def:process:OGC:detector"
            xlink:href="urn:ogc:sensor:Davis:7817"/>
        <!--~~~~~~~~~~~~~~~~~~-->
        <!--  Anemometer   -->
        <!--~~~~~~~~~~~~~~~~~~-->
        <component name="windSpeedTransducer" xlink:arcrole="urn:ogc:def:process:OGC:detector"
            xlink:href="urn:ogc:sensor:Davis:7911Spd"/>
```

```
            <component name="windDirectionTransducer" xlink:role="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:ogc:sensor:Davis:7911Dir"/>
            <!--~~~~~~~~~~~~~~~~~~-->
            <!--  Rain Gauge   -->
            <!--~~~~~~~~~~~~~~~~~~-->
            <component name="rainGaugeSensor" xlink:role="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:ogc:sensor:Davis:7852M"/>
            <!--~~~~~~~~~~~~~~~~~~-->
            <!--  Wind Chill   -->
            <!--~~~~~~~~~~~~~~~~~~-->
            <component name="windChill" xlink:role="urn:ogc:def:process:OGC:process"
                xlink:href="urn:ogc:process:windChill_v01"/>
        </ComponentList>
    </components>
```

The *System* definition provides a means to list, through the *positions* property, the relative positions of all components within the system, as well as the position of the *System* itself. Positions can also be provided by the *position* property of each component. Regardless, all positions relate the location and orientation of the local frame to an external reference frame. The local frame and reference frame should be specified using the *localFrame* and *referenceFrame* attributes within *Position* if they are the same for *location* and *orientation*. If the *location* and *orientation* are based on different reference frames, for example, then one should specify the *localFrame* and *referenceFrame* attributes of the respective *Vector* object instead.

Issue Name: SWE position versus GML location (meb; 2007-03-20)

Issue Description:

The SensorML encoding supports two means for providing positional data with physical processes (e.g. *Component* and *System*). The *location* property takes either a *gml:Point* or *gml:_Curve*. This is for compatibility with other GML *Features*.

However, within SWE models and encodings there is a slightly different view of position than that taken within GML:

(a) position is the result of a measurement and should thus be treated in the same manner as other measured properties in SWE

(b) position can provide full state of an object (i.e. location, orientation, and derivative properties such as velocities and accelerations) and not just location

(c) positional data is not a geometry but discrete time-dependent measurements from devices such as GPS and Inertial Navigation Systems

(d) Positional data needs to be linkable within SensorML process chains and systems in the same manner as other data and parameters

(e) Positional data relates one local coordinate reference system (CRS) to another external CRS. Thus, to be robust, positional data should state both the local AND the external reference CRS, and not simply the reference CRS as in GML.

For these reasons, the SWE community favors the use of *swe:Position* and

swe:Vector objects derived using SWE Common data types. The position property within *sml:Component* and *sml:System* takes either *swe:Position*, *swe:Vector*, or a *sml:_Process* for deriving the location. Examples of processes that provide position data include a GPS sensor, an orbital propagation process for satellites, or simply a look-up-table process for time-tagged positions.

Resolution:

In this example, the individual components are positioned and oriented relative to the station (or *System*) reference frame, as illustrated in Fig. 11.1.

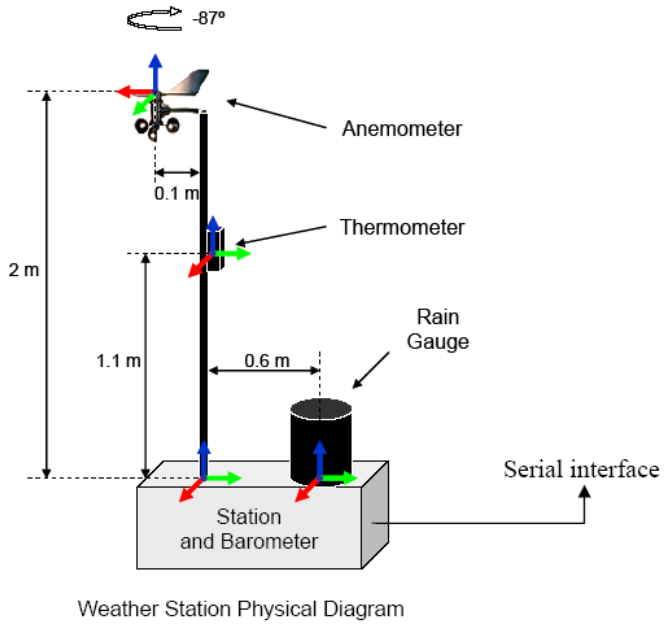


Figure 11.1. Positions of components of the weather station used within the System example. Locations and orientation are specified relative to the station (or System) spatial reference frame.

The first *position* in the example gives the position of the weather station itself relative to the geospatial coordinate reference system, urn:ogc:crs:EPSG:4329, which is the designation for latitude, longitude, and altitude based on the WGS84 world datum. All other spatial frames are specified by reference to spatial reference systems defined in each component. That is, "#THERMOMETER_FRAME" references the *gml:EngineeringCRS* (with *gml:id*="THERMOMETER_FRAME") of the thermometer that was defined in the example of Section 11.2.

```xml
<positions>
    <PositionList>
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
        <!-- Position of Station in Lat, Lon, Alt -->
        <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
        <position name="stationPosition">
            <swe:Position localFrame="#STATION_FRAME"
                            referenceFrame="urn:ogc:crs:EPSG:4329">
                <swe:location>
                    <swe:Vector gml:id="STATION_LOCATION"
                            definition="urn:ogc:def:property:OGC:location">
                        <swe:coordinate name="latitude">
                            <swe:Quantity axisID="Y">
                                <swe:uom code="deg"/>
                                <swe:value>34.72450</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="longitude">
                            <swe:Quantity axisID="X">
                                <swe:uom code="deg"/>
                                <swe:value>-86.94533</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="altitude">
                            <swe:Quantity axisID="Z">
                                <swe:uom code="m"/>
                                <swe:value>20.1169</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                    </swe:Vector>
                </swe:location>
                <swe:orientation>
                    <swe:Vector definition="urn:ogc:def:property:OGC:orientation">
                        <swe:coordinate name="trueHeading">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:angleToNorth"
                                axisID="Z">
                                <swe:uom code="deg"/>
                                <swe:value>87.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                    </swe:Vector>
                </swe:orientation>
            </swe:Position>
        </position>
        <!--==============================================-->
        <!-- Position of Barometer in Station Ref Frame -->
        <!--==============================================-->
        <position name="barometerPosition">
            <swe:Position localFrame="#BAROMETER_FRAME"
                referenceFrame="#STATION_FRAME">
                <swe:location>
                    <swe:Vector definition="urn:ogc:def:property:OGC:location">
                        <swe:coordinate name="x">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="X">
                                <swe:uom code="m"/>
                                <swe:value>0.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="y">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="Y">
                                <swe:uom code="m"/>
                                <swe:value>0.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
```

```
                        <swe:coordinate name="z">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="Z">
                                <swe:uom code="m"/>
                                <swe:value>0.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                    </swe:Vector>
                </swe:location>
            </swe:Position>
        </position>
        <!--============================================-->
        <!-- Position of Thermometer in Station Ref Frame -->
        <!--============================================-->
        <position name="thermometerPosition">
            <swe:Position localFrame="#THERMOMETER_FRAME"
                referenceFrame="#STATION_FRAME">
                <swe:location>
                    <swe:Vector definition="urn:ogc:def:property:OGC:location">
                        <swe:coordinate name="x">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="X">
                                <swe:uom code="m"/>
                                <swe:value>0.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="y">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="Y">
                                <swe:uom code="m"/>
                                <swe:value>0.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="z">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="Z">
                                <swe:uom code="m"/>
                                <swe:value>1.1</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                    </swe:Vector>
                </swe:location>
            </swe:Position>
        </position>
        <!--============================================-->
        <!-- Position of Anemometer in Station Ref Frame -->
        <!--============================================-->
        <position name="anemometerPosition">
            <swe:Position localFrame="#ANEMOMETER_FRAME"
                referenceFrame="#STATION_FRAME">
                <swe:location>
                    <swe:Vector definition="urn:ogc:def:property:OGC:location">
                        <swe:coordinate name="x">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="X">
                                <swe:uom code="m"/>
                                <swe:value>0.0</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="y">
                            <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                                axisID="Y">
                                <swe:uom code="m"/>
                                <swe:value>-0.1</swe:value>
                            </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="z">
```

```
                        <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                            axisID="Z">
                            <swe:uom code="m"/>
                            <swe:value>2.0</swe:value>
                        </swe:Quantity>
                    </swe:coordinate>
                </swe:Vector>
            </swe:location>
        </swe:Position>
    </position>
    <!--============================================================-->
    <!-- Position of Wind Direction Detector in Station Ref Frame -->
    <!--============================================================-->
    <position name="windDirectionDetectorPosition">
        <swe:Position localFrame="#WIND_DIRECTION_DETECTOR_FRAME"
            referenceFrame="#STATION_FRAME">
            <swe:location>
                <swe:Vector definition="urn:ogc:def:property:OGC:location">
                    <swe:coordinate name="x">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                            axisID="X">
                            <swe:uom code="m"/>
                            <swe:value>0.0</swe:value>
                        </swe:Quantity>
                    </swe:coordinate>
                    <swe:coordinate name="y">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                            axisID="Y">
                            <swe:uom code="m"/>
                            <swe:value>-0.1</swe:value>
                        </swe:Quantity>
                    </swe:coordinate>
                    <swe:coordinate name="z">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                            axisID="Z">
                            <swe:uom code="m"/>
                            <swe:value>2.0</swe:value>
                        </swe:Quantity>
                    </swe:coordinate>
                </swe:Vector>
            </swe:location>
            <swe:orientation>
                <swe:Vector definition="urn:ogc:def:property:OGC:orientation">
                    <swe:coordinate name="z">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:angle"
                            axisID="Z">
                            <swe:uom code="deg"/>
                            <swe:value>-87.0</swe:value>
                        </swe:Quantity>
                    </swe:coordinate>
                </swe:Vector>
            </swe:orientation>
        </swe:Position>
    </position>
    <!--============================================-->
    <!-- Position of Rain Gauge in Station Ref Frame -->
    <!--============================================-->
    <position name="rainGaugePosition">
        <swe:Position localFrame="#RAIN_GAUGE_FRAME"
            referenceFrame="#STATION_FRAME">
            <swe:location>
                <swe:Vector definition="urn:ogc:def:property:OGC:location">
                    <swe:coordinate name="x">
                        <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                            axisID="X">
                            <swe:uom code="m"/>
                            <swe:value>0.0</swe:value>
```

```
                    </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="y">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                        axisID="Y">
                        <swe:uom code="m"/>
                        <swe:value>0.6</swe:value>
                    </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="z">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:distance"
                        axisID="Z">
                        <swe:uom code="m"/>
                        <swe:value>0.0</swe:value>
                    </swe:Quantity>
                </swe:coordinate>
            </swe:Vector>
        </swe:location>
    </swe:Position>
</position>
</PositionList>
</positions>
```

The last property of a *System* is *connections*, which provides a list of all connections between *inputs*, *outputs*, and *parameters* within the *System*. The *Link* object in *System* follows the same rules as that in *ProcessChain*. Although the components in this example are referenced instead of described inline, it is important to note that links to components are specified the same whether the *component* is referenced using *xlink:href* or defined inline.

```
<connections>
    <ConnectionList>
        <connection name="inputToThermometer">
            <Link>
                <source ref="this/inputs/atmosphericTemperature"/>
                <destination ref="thermometer/inputs/temperature"/>
            </Link>
        </connection>
        <connection name="thermometerToOutput">
            <Link>
                <source ref="thermometer/outputs/measuredTemperature"/>
                <destination ref="this/outputs/weatherMeasurements/temperature"/>
            </Link>
        </connection>
        <connection name="inputToBarometer">
            <Link>
                <source ref="this/inputs/atmosphericPressure"/>
                <destination ref="barometer/inputs/barometricPressure"/>
            </Link>
        </connection>
        <connection name="barometerToOutput">
            <Link>
                <source ref="barometer/outputs/measuredBarometricPressure"/>
                <destination ref="this/outputs/weatherMeasurements/barometricPressure"/>
            </Link>
        </connection>
        <connection name="inputToAnemometer">
            <Link>
                <source ref="this/inputs/windSpeed"/>
                <destination ref="anemometer/inputs/windSpeed"/>
            </Link>
        </connection>
        <connection name="anemometerToOutput">
```

```
                    <Link>
                        <source ref="anemometer/outputs/measuredWindSpeed"/>
                        <destination ref="this/outputs/weatherMeasurements/windSpeed"/>
                    </Link>
                </connection>
                <connection name="inputToWindDirection">
                    <Link>
                        <source ref="this/inputs/windDirection"/>
                        <destination ref="windDirectionDetector/inputs/windDirection"/>
                    </Link>
                </connection>
                <connection name="windDirectionToOutput">
                    <Link>
                        <source ref="windDirectionDetector/outputs/measuredWindDirection"/>
                        <destination ref="this/outputs/weatherMeasurements/windDirection"/>
                    </Link>
                </connection>
                <connection name="inputToRainGauge">
                    <Link>
                        <source ref="this/inputs/rainFall"/>
                        <destination ref="rainGauge/inputs/rainFall"/>
                    </Link>
                </connection>
                <connection name="rainGaugeToOutput">
                    <Link>
                        <source ref="rainGauge/outputs/measuredRainFall"/>
                        <destination ref="this/outputs/weatherMeasurements/rainFall"/>
                    </Link>
                </connection>
                <connection name="clockToOutput">
                    <Link>
                        <source ref="clock/outputs/measuredTime"/>
                        <destination ref="this/outputs/weatherMeasurements/time"/>
                    </Link>
                </connection>
            </ConnectionList>
        </connections>
    </System>
    <!-- ~~~~~~~~~~ End of System ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  </member>
</SensorML>
```

10.5 Metadata Group

The SensorML *metadataGroup* provides a common collection of metadata elements that can be used within any process or process method definition.

> **Issue Name**: Harmonization of Metadata schema with ISO TC211. (meb, 2007-01-08)
>
> **Issue Description:** Many of the metadata components in SensorML are based on models from ISO TC211 (particularly ISO19115), including *sml:ResponsibleParty*, *sml:Contact*, *sml:Address*, and *sml:LegalConstraints*. There is an ongoing effort in ISO to create standard XML Schema for these metadata elements within the *gmd* namespace. These ISO schema are not directly utilized within SensorML V1.0.
>
> **Resolution:** It is anticipated that SensorML V1.1 will directly and more fully utilize the ISO schemas being developed.

10.5.1 Keywords, Identification, and Classification

The *keywords*, *identifier* and *classifier* properties provide both human and software readable names, types, and applications of resources. Both *identifier* and *classifier* take a *Term* which has an optional *definition* attribute that further refines the meaning of the token value. The optional *codeSpace* element further refines the value of the *Term* or *KeywordList* by referencing a dictionary or ontology that defines choices or rules for the value.

A *definition* value for an *identifier* might include, for example, a short name for use perhaps in menus and data trees, a long name, mission ID, or model number. Examples of *definition* attribute values for c*lassifier* might include sensor type or application. The *identification* and *classification* properties provide relevant information that can be mined to support asset discovery and cataloging.

An example for the Davis thermometer is given below:

```
<keywords>
    <KeywordList codeSpace="urn:x-nasa:def:gcmd:keywords">
        <keyword>weather</keyword>
        <keyword>insitu</keyword>
        <keyword>station</keyword>
    </KeywordList>
</keywords>
<identification>
    <IdentifierList>
        <identifier name="longName">
            <Term definition="urn:ogc:def:identifier:OGC:longname">
                <value>Davis 7817 External Temperature Sensor</value>
            </Term>
        </identifier>
        <identifier name="shortName">
            <Term definition="urn:ogc:def:identifier:OGC:shortname">
                <value>Davis 7817 Thermometer</value>
            </Term>
        </identifier>
        <identifier name="modelNumber">
            <Term definition="urn:ogc:def:identifier:OGC:modelNumber">
                <codeSpace xlink:href="urn:x-davisweather:sensors:models"/>
                <value>7817</value>
            </Term>
        </identifier>
        <identifier name="manufacturer">
            <Term definition="urn:ogc:def:identifier:OGC:manufacturer">
                <value>Davis Instruments</value>
            </Term>
        </identifier>
    </IdentifierList>
</identification>
<classification>
    <ClassifierList>
        <classifier name="intendedApplication">
            <Term definition="urn:ogc:def:classifier:OGC:application">
                <value>weather</value>
            </Term>
        </classifier>
        <classifier name="sensorType">
            <Term definition="urn:sensor:classifier:sensorType">
                <codeSpace xlink:href="urn:x-ceos:def:GCMD:sensors"/>
                <value>thermometer</value>
            </Term>
        </classifier>
    </ClassifierList>
```

```
</classification>
```

10.5.2 Constraints

Any SensorML document can be constrained by time, or by security and legal constraints. If a physical process in SensorML has a *validTime* property, the document should be interpreted as defining the configuration of the Component or System over that operational time period.

The security constraints are provided by the *securityConstraints* property and are specified as a set of attributes defined using the United States Intelligence Community's ISM standard. For further information about these attributes, one should reference that standard. The *legalConstraints* property provides a means to declare copyrights, intellectual property, licenses, liability restrictions, or privacy rights declarations, and allows reference to associated documents and responsible parties.

```
<validTime>
    <gml:TimePeriod gml:id="ValidTime">
        <gml:beginPosition xlink:arcrole="urn:ogc:def:property:OGC:deployment">
            2005-10-20T20:00:00Z
        </gml:beginPosition>
        <gml:endPosition indeterminatePosition="now"/>
    </gml:TimePeriod>
</validTime>
<securityConstraint>
    <Security ism:classification="U"/>
</securityConstraint>
<legalConstraint>
    <Rights>
        <documentation xlink:arcrole="urn:ogc:def:role:liabilities">
            <Document>
                <gml:description>
                    The author of this document is not responsible for
                    the accuracy of the information, nor for anything
                    whatsoever
                </gml:description>
                <onlineResource xlink:href="http://www.bobbyRayLaw.com/notMe.html"/>
            </Document>
        </documentation>
    </Rights>
</legalConstraint>
```

10.5.3 Characteristics and Capabilities

The *characteristics* and *capabilities* properties provide information that is useful for discovery and for assistance to humans. Both take any element derived from *AbstractDataRecord*, but would typically be of type *DataRecord*. Characteristics and capabilities are defined using the SWE Common data types, typically *Quantity*, *Category*, *Boolean*, *Count*, and *DataRecord*. Typical characteristics could include physical properties or power requirements, while capabilities might include measurement characteristics or operational limits. However, they could include any properties as defined by the *definition* attribute. It is envisioned that profiles of capabilities and characteristics will be defined for particular classes of processes and sensors.

The following example is from the Davis external thermometer:

```
<characteristics>
    <swe:DataRecord definition="urn:ogc:def:property:OGC:physicalProperties">
        <swe:field name="physicalProperties">
            <swe:DataRecord>
                <swe:field name="weight">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:weight">
                        <swe:uom code="g"/>
                        <swe:value>128</swe:value>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="diameter">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:diameter">
                        <swe:uom code="mm"/>
                        <swe:value>6.5</swe:value>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="length">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:length">
                        <swe:uom code ="mm"/>
                        <swe:value>32</swe:value>
                    </swe:Quantity>
                </swe:field>
            </swe:DataRecord>
        </swe:field>
    </swe:DataRecord>
</characteristics>
<capabilities>
    <swe:DataRecord definition="urn:ogc:def:property:OGC:measurementProperties">
        <gml:description>
            The External Temperature Sensor is used to measure temperatures in general
            conditions. It is well-suited for air, water, or soil temperature measurements,
            and it may be used anywhere a reliable, low-cost temperature sensor is required.
            The sensor is epoxy-encapsulated in a vinyl cap. The External Temperature Sensor
            uses a precision platinum wire thermistor as a sensor. The thermistor produces a
            resistance change proportional to temperature. To ensure accurate readings when
            measuring outdoor air temperature, the External Temperature Sensor should be
            shielded from direct sunlight and other sources of reflected or radiated heat.
            We recommend the use of a Davis Radiation Shield (#7714) or its equivalent for
            this purpose.</gml:description>
        <!-- add EnvironmentalLimit group -->
        <swe:field name="resolution" xlink:arcrole="urn:ogc:def:property:OGC:resolution">
            <swe:Quantity definition="urn:ogc:def:property:OGC:temperature">
                <swe:uom code="cel"/>
                <swe:value>0.1</swe:value>
            </swe:Quantity>
        </swe:field>
        <swe:field name="range" xlink:arcrole="urn:ogc:def:property:OGC:dynamicRange">
            <swe:QuantityRange definition="urn:ogc:def:property:OGC:temperature">
                <swe:uom code="cel"/>
                <swe:value>-45 60</swe:value>
            </swe:QuantityRange>
        </swe:field>
        <swe:field name="accuracy" xlink:arcrole="urn:ogc:def:property:OGC:accuracy">
            <swe:QuantityRange definition="urn:ogc:def:property:OGC:absoluteAccuracy">
                <swe:uom xlink:href="urn:ogc:unit:percent"/>
                <swe:value>-0.5 0.5</swe:value>
            </swe:QuantityRange>
        </swe:field>
    </swe:DataRecord>
</capabilities>
```

10.5.4 References

The *references* group provides the ability to specify persons or responsible parties, as well as relevant documentation. The *contact* property accepts either a *ResponsibleParty* (based on ISO 19115), a *Person* (based on the IC:DMMS standard), or a *ContactList* as its value. The role of the contact with respect to this object (e.g. expert, manufacturer, operator, etc) should be stated using the *xlink:arcrole* attribute within the *contact* association element, as in the following example:

```
<contact xlink:arcrole="urn:ogc:def:role:manufacturer">
  <ResponsibleParty >
    <organizationName>Davis Instruments</organizationName>
    <contactInfo>
        <phone>
            <voice>+01-510-732-9229</voice>
            <facsimile>+01-510-732-9188</facsimile>
        </phone>
        <address>
            <deliveryPoint>3465 Diablo Avenue</deliveryPoint>
            <city>Hayward</city>
            <administrativeArea>CA</administrativeArea>
            <postalCode>94545-2778</postalCode>
            <country>USA</country>
            <electronicMailAddress>sales@davisnet.com</electronicMailAddress>
        </address>
    </contactInfo>
  </ResponsibleParty>
</contact>
```

The *documentation* property takes a *Document* (based on ISO 19115) or *DocumentList* as its value. The role of the document with respect to the enclosing object (e.g. specification sheet, image of object, algorithm description, etc.) should be provided using the *xlink:arcrole* attribute of the *documentation* element.

```
    <documentation xlink:arcrole="urn:ogc:role:specificationSheet">
        <Document>
            <gml:description>specification sheet</gml:description>
            <format>pdf</format>
            <onlineResource
xlink:href="http://www.davisnet.com/product_documents/weather/spec_sheets/ext_temp_sensor_std.pdf"/>
        </Document>
    </documentation>
    <documentation xlink:arcrole="urn:ogc:def:role:objectImage">
        <Document>
            <gml:description>image of the external thermometer 7817</gml:description>
            <format>/mime/image/jpg</format>
            <onlineResource xlink:href="http://www.davisnet.com/productpics/big/7817.jpg"/>
        </Document>
    </documentation>
```

10.5.5 History

The *history* property allows for one or more events to be described inline or referenced externally. Such events could include deployment, calibration, repairs, algorithm corrections, and decommissioning, to name a few. The roles for the events should be specified using the *xlink:arcrole* attribute of the *history* or *member* properties. It is

anticipated but not required that history will typically be maintained online in an external document and referenced from within the process description.

```
<history>
    <EventList>
        <member name="deployment" xlink:arcrole="urn:ogc:def:property:OGC:deployment">
            <Event>
                <date>2004-02-10T10:00:00Z</date>
                <gml:description>Deployment event</gml:description>
                <contact xlink:arcrole="operator" xlink:href="http://www.myCompany.com/bob.xml"/>
                <documentation xlink:arcrole="deploymentNotes">
                    <Document>
                        <gml:description>deployment report</gml:description>
                        <onlineResource xlink:href="http://www.myCompany.com/bobsNotes.pdf"/>
                    </Document>
                </documentation>
            </Event>
        </member>
        <member name="calibration-2005-10-03"
xlink:arcrole="urn:ogc:def:property:OGC:calibration">
            <Event gml:id="cal0287365">
                <date>2005-10-03T14:00:00Z</date>
                <gml:description>calibration event</gml:description>
                <contact xlink:arcrole="operator" xlink:href="http://www.myCompany.com/jeff.xml"/>
                <documentation xlink:arcrole="calibrationReport">
                    <Document>
                        <gml:description>Calibration report and data</gml:description>
                        <onlineResource
                        xlink:href="http://www.myCompany.com/calibration_0023_2005-10-03.xml"/>
                    </Document>
                </documentation>
            </Event>
        </member>
    </EventList>
</history>
```

10.6 SensorML Profiles

The desire to create application-specific profiles from SensorML is anticipated. Typically this would be accomplished by either deriving application schema from SensorML by restriction, by the use of RelaxNG, Schematron, XML Schema, or a combination of the these. Other means of providing finer-grained profiling have been developed within the industry. All current technology for defining and validating fine-grained XML profiles provide both advantages and limitations.

Best practices for profiling SensorML is beyond the scope of this document, but will be addressed in separate documentation.

11 Future Directions and Remaining Issues

It is anticipated that there will be both minor refinements and additions to the existing schema in future versions and as part of SensorML extensions. Furthermore, efforts are underway to implement software capable of parsing SensorML, mining information for catalogs, processing and geolocating observation data based on SensorML instances, displaying SensorML instances in a human-friendly viewer, and creating SensorML instances using graphical interfaces. These will provide lessons learned that will be used to refine the schemas of SensorML.

Below is a limited list of recognized desires for advancement of SensorML and needs in future releases.

1. Establish authoritative dictionaries and ontologies process and sensor-related terms.
2. Begin development of common *ProcessModel* definitions complete with method descriptions and software code.
3. Develop and document Best Practices for fine-grained profiling of sensors and processes.
4. Finish development implementation within SensorML of standard sensor models for geolocation of observations from remote sensors. Support both rigorous and polynomial models. Work with US Community Sensor Model Working Group and ISO 19130 projects.
5. Investigate the application of MathML for SensorML method descriptions and execution.
6. Investigate avenues and implement solutions for increased synergy between SensorML and GML. This may involve some slight modifications to both SensorML and GML.
7. Provide mapping between SensorML and other sensor standards, such IEEE P1451 and ANSI N42.42.
8. Continue harmonization efforts of TransducerML with SWE Common, Observations, and SensorML.
9. Investigate and implement the use of SensorML process descriptions within industry-standard distributed processing protocols (e.g., BPEL). In this regard, we recognize SensorML as a potential description language for defining any process or process chain independent of the means or protocol by which the process has or will be executed.

Annex A. XML Schemas for SensorML (normative)

A.1 base.xsd.

The base.xsd schema provides the basic definitions and abstract elements used by several SensorML schema. It also defines the metadata groups and components.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/sensorML/1.0"
    xmlns:ism="urn:us:gov:ic:ism:v2"
    xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--  -->
    <xs:annotation>
        <xs:documentation>Base class definitions for core SensorML</xs:documentation>
    </xs:annotation>
    <!-- ================================================ -->
    <!-- import GML 3.1.1 -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <!-- import SWE Common Data Types 1.0.0 -->
    <xs:import namespace="http://www.opengis.net/swe/1.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/1.0.0/swe.xsd"/>
    <!-- import US Intelligence Community schema for security speciifications-->
    <xs:import namespace=" urn:us:gov:ic:ism:v2"
        schemaLocation=" http://schemas.opengis.net /ic/2.0/IC-ISM-v2.xsd"/>
    <!-- ================================================ -->
    <xs:complexType name="AbstractSMLType" abstract="true">
        <xs:annotation>
            <xs:documentation>Main Abstract SensorML Object</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:restriction base="gml:AbstractFeatureType">
                <xs:sequence>
                    <xs:sequence>
                        <xs:sequence>
                            <xs:element ref="gml:description" minOccurs="0"/>
                            <xs:element ref="gml:name" minOccurs="0">
                                <xs:annotation>
                                    <xs:documentation>
                                    The gml:name property should be used only as a label in SensorML, and thus
                                    is limited to one occurence. Multiple robust and well defined identifiers can be
                                    provided in the identification section.
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element ref="gml:boundedBy" minOccurs="0">
                                <xs:annotation>
                                    <xs:documentation>
                                        Specifies the possible extent of the component location
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:element>
                        </xs:sequence>
                    </xs:sequence>
                </xs:sequence>
            </xs:restriction>
```

```
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================== -->
    <xs:group name="constraints">
        <xs:sequence>
            <xs:element ref="sml:validTime" minOccurs="0"/>
            <xs:element ref="sml:securityConstraint" minOccurs="0"/>
            <xs:element ref="sml:legalConstraint" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:group>
    <!-- ================================================== -->
    <xs:element name="securityConstraint">
        <xs:annotation>
            <xs:documentation>Means of providing security constraints of description</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="sml:Security"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <xs:element name="Security">
        <xs:annotation>
            <xs:documentation>based on IC:ISM definition</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:attributeGroup ref="ism:SecurityAttributesOptionGroup"/>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <xs:element name="validTime">
        <xs:annotation>
            <xs:documentation>Means of providing time validity constraint of description</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice>
                <xs:element ref="gml:TimeInstant"/>
                <xs:element ref="gml:TimePeriod"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <xs:element name="legalConstraint">
        <xs:annotation>
            <xs:documentation>Means of providing legal constraints of description</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence minOccurs="0">
                <xs:element ref="sml:Rights"/>
            </xs:sequence>
            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <xs:element name="Rights">
        <xs:annotation>
            <xs:documentation>based on IC:DDMS definition</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="sml:documentation"/>
            </xs:sequence>
            <xs:attribute ref="gml:id"/>
            <xs:attribute name="privacyAct" type="xs:boolean" use="optional"/>
            <xs:attribute name="intellectualPropertyRights" type="xs:boolean" use="optional"/>
            <xs:attribute name="copyRights" type="xs:boolean" use="optional"/>
```

```
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:group name="generalInfo">
    <xs:sequence>
        <xs:element ref="sml:keywords" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="sml:identification" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="sml:classification" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:group>
<!-- ================================================== -->
<xs:element name="keywords">
    <xs:annotation>
        <xs:documentation>
            Means of providing a list of keywords (with a codeSpace) for quick discovery
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="KeywordList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="keyword" type="xs:token" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:ID"/>
                    <xs:attribute name="codeSpace" type="xs:anyURI"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="identification">
    <xs:annotation>
        <xs:documentation>
            Means of providing various identity and alias values, with types such as "longName",
            "abbreviation", "modelNumber", "serialNumber", whose terms can be defined in a
            dictionary
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="IdentifierList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="identifier" maxOccurs="unbounded">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="sml:Term"/>
                                </xs:sequence>
                                <xs:attribute name="name" type="xs:token" use="optional"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:ID"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="classification">
    <xs:annotation>
        <xs:documentation>
            Means of specifying classification values with types such as "sensorType", "intendedApplication",
```

```
                etc., whose terms can be defined in a dictionary
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence minOccurs="0">
                <xs:element name="ClassifierList">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="classifier" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element ref="sml:Term"/>
                                    </xs:sequence>
                                    <xs:attribute name="name" type="xs:token" use="optional"/>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:ID"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <xs:element name="Term">
        <xs:annotation>
            <xs:documentation>
                A well defined token used to specify identifier and classifier values (single spaces allowed)
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="codeSpace" type="swe:CodeSpacePropertyType" minOccurs="0"/>
                <xs:element name="value" type="xs:token"/>
            </xs:sequence>
            <xs:attribute name="definition" type="xs:anyURI">
                <xs:annotation>
                    <xs:documentation>
                        Points to the term definition using a URI. Term definitions are things like uid, shortName,
                        sensorType, application, etc...
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <xs:group name="references">
        <xs:sequence>
            <xs:element ref="sml:contact" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="sml:documentation" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:group>
    <!-- ================================================== -->
    <xs:element name="contact">
        <xs:annotation>
            <xs:documentation>Relevant contacts for that object</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice minOccurs="0">
                <xs:group ref="sml:ContactGroup"/>
                <xs:element ref="sml:ContactList"/>
            </xs:choice>
            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
```

```
<xs:element name="ContactList">
    <xs:annotation>
        <xs:documentation>
            Allows to group several contacts together in a list that can be referenced as a whole
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="gml:description" minOccurs="0"/>
            <xs:element name="member" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:group ref="sml:ContactGroup"/>
                    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute ref="gml:id"/>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:group name="ContactGroup">
    <xs:choice>
        <xs:element ref="sml:Person"/>
        <xs:element ref="sml:ResponsibleParty"/>
    </xs:choice>
</xs:group>
<!-- ================================================ -->
<xs:element name="Person">
    <xs:annotation>
        <xs:documentation>based on IC:DMMS</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="surname" type="xs:token"/>
            <xs:element name="name" type="xs:token"/>
            <xs:element name="userID" type="xs:token"/>
            <xs:element name="affiliation" type="xs:token"/>
            <xs:element name="phoneNumber" type="xs:token"/>
            <xs:element name="email" type="xs:token"/>
        </xs:sequence>
        <xs:attribute ref="gml:id"/>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:element name="ResponsibleParty">
    <xs:annotation>
        <xs:documentation>based on ISO 19115</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="individualName" type="xs:string" minOccurs="0"/>
            <xs:element name="organizationName" type="xs:string" minOccurs="0"/>
            <xs:element name="positionName" type="xs:string" minOccurs="0"/>
            <xs:element ref="sml:contactInfo" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute ref="gml:id"/>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:element name="contactInfo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="phone" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="voice" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="facsimile" type="xs:string" minOccurs="0"
```

```
                        maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="address" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="deliveryPoint" type="xs:string" minOccurs="0"
                        maxOccurs="unbounded"/>
                    <xs:element name="city" type="xs:string" minOccurs="0"/>
                    <xs:element name="administrativeArea" type="xs:string" minOccurs="0"/>
                    <xs:element name="postalCode" type="xs:string" minOccurs="0"/>
                    <xs:element name="country" type="xs:string" minOccurs="0"/>
                    <xs:element name="electronicMailAddress" type="xs:string" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element ref="sml:onlineResource" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="hoursOfService" type="xs:string" minOccurs="0"/>
        <xs:element name="contactInstructions" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-- ============================================== -->
<xs:element name="onlineResource">
  <xs:complexType>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
  </xs:complexType>
</xs:element>
<!-- ============================================== -->
<xs:element name="documentation">
  <xs:annotation>
    <xs:documentation>Relevant documentation for that object</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:choice minOccurs="0">
      <xs:element ref="sml:Document"/>
      <xs:element ref="sml:DocumentList"/>
    </xs:choice>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
  </xs:complexType>
</xs:element>
<!-- ============================================== -->
<xs:element name="Document">
  <xs:annotation>
    <xs:documentation>
      Document record with date/time, version, author, etc. pointing to an online resource related to the
      enclosing object
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="gml:description"/>
      <xs:element name="date" type="swe:timeIso8601" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Date of creation</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element ref="sml:contact" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Person who is responsible for the document</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="format" type="xs:token" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            Specifies the fornat of the file pointed to by onlineResource
```

```
                    </xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element ref="sml:onlineResource" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>
                        Points to the actual document corresponding to that version
                    </xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
        <xs:attribute ref="gml:id"/>
        <xs:attribute name="version" type="xs:token" use="optional"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="DocumentList">
    <xs:annotation>
        <xs:documentation>List of documents related to the enclosing object</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="gml:description" minOccurs="0"/>
            <xs:element name="member" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence minOccurs="0">
                        <xs:element ref="sml:Document"/>
                    </xs:sequence>
                    <xs:attribute name="name" type="xs:token" use="required"/>
                    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute ref="gml:id"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:group name="history">
    <xs:sequence>
        <xs:element ref="sml:history" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:group>
<!-- ================================================== -->
<xs:element name="history">
    <xs:annotation>
        <xs:documentation>
            History of the object described (Recalibration, adjustments, etc...)
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element ref="sml:EventList"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="EventList">
    <xs:annotation>
        <xs:documentation>List of events related to the enclosing object</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="member" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence minOccurs="0">
                        <xs:element ref="sml:Event"/>
```

```
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:token" use="required"/>
                        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute ref="gml:id"/>
        </xs:complexType>
    </xs:element>
    <!-- ============================================== -->
    <xs:element name="Event">
        <xs:annotation>
            <xs:documentation>
                Event record (change to the object) including a date/time, description, identification and additional
                references and metadata
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="date" type="swe:timeIso8601" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Date/Time of event</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element ref="gml:description" minOccurs="0"/>
                <xs:group ref="sml:generalInfo" minOccurs="0"/>
                <xs:group ref="sml:references" minOccurs="0"/>
                <xs:element name="property" type="swe:DataComponentPropertyType" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="gml:id"/>
        </xs:complexType>
    </xs:element>
    <!-- ============================================== -->
    <xs:group name="properties">
        <xs:sequence>
            <xs:element ref="sml:characteristics" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="sml:capabilities" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:group>
    <!-- ============================================== -->
    <xs:element name="capabilities">
        <xs:annotation>
            <xs:documentation>Capability list for quick discovery</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence minOccurs="0">
                <xs:element ref="swe:AbstractDataRecord"/>
            </xs:sequence>
            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
        </xs:complexType>
    </xs:element>
    <!-- ============================================== -->
    <xs:element name="characteristics">
        <xs:annotation>
            <xs:documentation>Characteristic list for quick discovery</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence minOccurs="0">
                <xs:element ref="swe:AbstractDataRecord"/>
            </xs:sequence>
            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

A.2 process.xsd.

The process.xsd schema provides definitions for base process types, as well as concrete definitions for ProcessChain and ProcessModel.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/sensorML/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--  -->
    <xs:annotation>
        <xs:documentation>Defines Basic Process Elements and Types for SensorML</xs:documentation>
    </xs:annotation>
    <!-- ================================================= -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:import namespace="http://www.opengis.net/swe/1.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/1.0.0/swe.xsd"/>
    <xs:include schemaLocation="./method.xsd"/>
    <!-- ================================================= -->
    <xs:element name="_Process" type="sml:AbstractProcessType" abstract="true">
        <xs:annotation>
            <xs:documentation>base substitution group for all processes</xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ================================================= -->
    <xs:complexType name="AbstractProcessType" abstract="true">
        <xs:complexContent>
            <xs:extension base="sml:AbstractSMLType">
                <xs:sequence>
                    <xs:group ref="sml:metadataGroup" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================= -->
    <xs:group name="metadataGroup">
        <xs:annotation>
            <xs:documentation>Group containing all metadata information</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:group ref="sml:generalInfo" minOccurs="0"/>
            <xs:group ref="sml:constraints" minOccurs="0"/>
            <xs:group ref="sml:properties" minOccurs="0"/>
            <xs:group ref="sml:references" minOccurs="0"/>
            <xs:group ref="sml:history" minOccurs="0"/>
        </xs:sequence>
    </xs:group>
    <!-- ================================================= -->
    <xs:complexType name="AbstractRestrictedProcessType" abstract="true">
        <xs:complexContent>
            <xs:restriction base="sml:AbstractProcessType">
                <xs:sequence>
                    <xs:sequence>
                        <xs:element ref="gml:description" minOccurs="0"/>
                        <xs:element ref="gml:name" minOccurs="0"/>
                    </xs:sequence>
                    <xs:group ref="sml:metadataGroup" minOccurs="0"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
```

```
<!-- ================================================ -->
<xs:complexType name="AbstractPureProcessType" abstract="true">
    <xs:annotation>
        <xs:documentation>Complex Type for all soft-typed processes</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="sml:AbstractRestrictedProcessType">
            <xs:sequence>
                <xs:element ref="sml:inputs" minOccurs="0"/>
                <xs:element ref="sml:outputs" minOccurs="0"/>
                <xs:element ref="sml:parameters" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="inputs">
    <xs:annotation>
        <xs:documentation>list of input signals</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="InputList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="input" type="sml:IoComponentPropertyType"
                            maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:ID"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:element name="outputs">
    <xs:annotation>
        <xs:documentation>list of output signals</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="OutputList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="output" type="sml:IoComponentPropertyType"
                            maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:ID"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:element name="parameters">
    <xs:annotation>
        <xs:documentation>list of parameters</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="ParameterList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="parameter" type="swe:DataComponentPropertyType"
                            maxOccurs="unbounded"/>
```

```
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:ID"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:element name="ProcessModel" type="sml:ProcessModelType" substitutionGroup="sml:_Process">
    <xs:annotation>
        <xs:documentation>Simple atomic process defined using a ProcessMethod</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="ProcessModelType">
    <xs:annotation>
        <xs:documentation>Complex Type for atomic processes</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="sml:AbstractPureProcessType">
            <xs:sequence>
                <xs:element ref="sml:method"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="ProcessChain" type="sml:ProcessChainType" substitutionGroup="sml:_Process">
    <xs:annotation>
        <xs:documentation>Process formed by chaining sub-processes</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="ProcessChainType">
    <xs:annotation>
        <xs:documentation>Complex Type for process chains</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="sml:AbstractPureProcessType">
            <xs:sequence>
                <xs:element ref="sml:components"/>
                <xs:element ref="sml:connections"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="components">
    <xs:annotation>
        <xs:documentation>
            Collection of subprocesses that can be chained using connections
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="ComponentList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="component" maxOccurs="unbounded">
                            <xs:complexType>
                                <xs:sequence minOccurs="0">
                                    <xs:element ref="sml:_Process"/>
                                </xs:sequence>
                                <xs:attribute name="name" type="xs:token" use="required"/>
                                <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
```

```
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ============================================== -->
<xs:element name="DataSource" type="sml:DataSourceType" substitutionGroup="sml:_Process">
    <xs:annotation>
        <xs:documentation>
            Process with no inputs representing a source of data (Tables, Observations...) for other processes
            to connect to.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="DataSourceType">
    <xs:complexContent>
        <xs:extension base="sml:AbstractProcessType">
            <xs:sequence>
                <xs:choice>
                    <xs:sequence>
                        <xs:element name="dataDefinition">
                            <xs:complexType>
                                <xs:choice minOccurs="0">
                                    <xs:element ref="swe:DataBlockDefinition"/>
                                    <xs:element ref="swe:DataStreamDefinition"/>
                                </xs:choice>
                                <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="values">
                            <xs:complexType>
                                <xs:complexContent>
                                    <xs:extension base="xs:anyType"/>
                                </xs:complexContent>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                    <xs:element name="observationReference">
                        <xs:complexType>
                            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                        </xs:complexType>
                    </xs:element>
                </xs:choice>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ============================================== -->
<xs:element name="connections">
    <xs:annotation>
        <xs:documentation>
            provides links between processes or between data sources and processes
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="ConnectionList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="sml:connection" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<!-- ================================================= -->
<xs:element name="connection">
    <xs:annotation>
        <xs:documentation>Specify a connection between two elements</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:choice>
            <xs:element ref="sml:Link"/>
            <xs:element ref="sml:ArrayLink"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:token" use="optional"/>
    </xs:complexType>
</xs:element>
<!-- ================================================= -->
<xs:element name="Link">
    <xs:annotation>
        <xs:documentation>Link object used to make connections between processes</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="source">
                <xs:complexType>
                    <xs:attribute name="ref" type="sml:linkRef" use="required"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="destination">
                <xs:complexType>
                    <xs:attribute name="ref" type="sml:linkRef" use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="type" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>
<!-- ================================================= -->
<xs:simpleType name="linkRef">
    <xs:restriction base="xs:token"/>
</xs:simpleType>
<!-- ================================================= -->
<xs:element name="ArrayLink">
    <xs:annotation>
        <xs:documentation>Special Link to handle accessing array elements sequentially</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:choice>
                <xs:sequence>
                    <xs:element name="sourceArray">
                        <xs:complexType>
                            <xs:attribute name="ref" type="sml:linkRef"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="destinationIndex" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:attribute name="ref" type="sml:linkRef"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:sequence>
                    <xs:element name="destinationArray">
                        <xs:complexType>
                            <xs:attribute name="ref" type="sml:linkRef"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="sourceIndex" minOccurs="0">
                        <xs:complexType>
                            <xs:attribute name="ref" type="sml:linkRef"/>
```

```
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:choice>
            <xs:element ref="sml:connection" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<!--
    NOTE: we intend to support switch between process using special ProcessModels rather than a
        special Link because it provides more flexibility (though it is more verbose)
-->
<!-- ================================================================== -->
<!-- Complex Types used for deriving hard-typed processes by restriction -->
<!-- ================================================================== -->
<xs:complexType name="AbstractDerivableProcessType" abstract="true">
    <xs:complexContent>
        <xs:extension base="sml:AbstractProcessType">
            <xs:sequence>
                <xs:element name="inputs" type="sml:inputsPropertyType" minOccurs="0"/>
                <xs:element name="outputs" type="sml:outputsPropertyType" minOccurs="0"/>
                <xs:element name="parameters" type="sml:parametersPropertyType" minOccurs="0"/>
                <xs:choice minOccurs="0">
                    <xs:element ref="sml:method"/>
                    <xs:sequence>
                        <xs:element name="components" type="sml:componentsPropertyType"/>
                        <xs:element name="dataSources" type="sml:dataSourcesPropertyType"
                            minOccurs="0"/>
                        <xs:element name="connections" type="sml:connectionsPropertyType"/>
                    </xs:sequence>
                </xs:choice>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="AbstractListType">
    <xs:complexContent>
        <xs:extension base="xs:anyType">
            <xs:attribute name="id" type="xs:ID" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="inputsPropertyType">
    <xs:sequence minOccurs="0">
        <xs:element name="InputList" type="sml:AbstractListType"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="outputsPropertyType">
    <xs:sequence minOccurs="0">
        <xs:element name="OutputList" type="sml:AbstractListType"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="parametersPropertyType">
    <xs:sequence minOccurs="0">
        <xs:element name="ParameterList" type="sml:AbstractListType"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="componentsPropertyType">
    <xs:sequence minOccurs="0">
```

```
            <xs:element name="ProcessList" type="sml:AbstractListType"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:complexType name="dataSourcesPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element name="DataSourceList" type="sml:AbstractListType"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:complexType name="connectionsPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element name="ConnectionList" type="sml:AbstractListType"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:complexType name="IoComponentPropertyType">
        <xs:choice minOccurs="0">
            <xs:group ref="swe:AnyData"/>
            <xs:element ref="swe:ObservableProperty"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:token" use="required"/>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:schema>
```

A.3 method.xsd.

The method.xsd schema a description of the process methodology including algorithms, documentation, fine-grained validation, and links to software implementations.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/sensorML/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--  -->
    <xs:annotation>
        <xs:documentation>Defines Process Method</xs:documentation>
    </xs:annotation>
    <!-- ============================================== -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:import namespace="http://www.opengis.net/swe/1.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/1.0.0/swe.xsd"/>
    <xs:include schemaLocation="./base.xsd"/>
    <xs:include schemaLocation="./process.xsd"/>
    <!-- ============================================== -->
    <xs:element name="ProcessMethod" type="sml:ProcessMethodType">
        <xs:annotation>
            <xs:documentation>Method describing a process (Can also be a dictionary entry)</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ProcessMethodType">
        <xs:annotation>
            <xs:documentation>Complex Type for process methods definition</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="gml:AbstractGMLType">
                <xs:sequence>
```

```
<xs:group ref="sml:metadataGroup" minOccurs="0"/>
<xs:element name="rules">
    <xs:annotation>
        <xs:documentation>
            Text and/or language defining rules for process profile (e.g. inputs, outputs,
            parameters, and metadata)
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="RulesDefinition">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="gml:description" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>Textual description of the i/o structure
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element ref="sml:ruleLanguage" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="algorithm" minOccurs="0">
    <xs:annotation>
        <xs:documentation>
            Textual and/or MathML description of the algorithm
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="AlgorithmDefinition">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="gml:description" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Textual description of the algorithm
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="mathML" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Includes or reference a MathML doc specifying the math of
                                    the algorithm
                                </xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                                <xs:sequence minOccurs="0">
                                    <xs:any namespace="##any" processContents="lax"/>
                                </xs:sequence>
                                <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="implementation" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>
```

```
                        Points to the reference implementation of this process in the specified
                        programming language (can be a SensorML process chain)
                    </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:choice minOccurs="0">
                        <xs:element ref="sml:ProcessChain"/>
                        <xs:element name="ImplementationCode">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="gml:description" minOccurs="0">
                                        <xs:annotation>
                                            <xs:documentation>
                                                Textual description of the algorithm
                                            </xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:group ref="sml:metadataGroup" minOccurs="0"/>
                                    <xs:element name="sourceRef" minOccurs="0">
                                        <xs:complexType>
                                            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="binaryRef" minOccurs="0">
                                        <xs:complexType>
                                            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                                <xs:attribute name="language" type="xs:token" use="required"/>
                                <xs:attribute name="framework" type="xs:token" use="optional"/>
                                <xs:attribute name="version" type="xs:token" use="optional"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- ============================================== -->
<xs:element name="method" type="sml:methodPropertyType">
    <xs:annotation>
        <xs:documentation>process method</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="methodPropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="sml:ProcessMethod"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ============================================== -->
<xs:element name="ruleLanguage" type="sml:ruleLanguageType" abstract="true">
    <xs:annotation>
        <xs:documentation>substitutionGroup for languages that define rules</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="ruleLanguageType">
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<xs:element name="schematron" substitutionGroup="sml:ruleLanguage">
    <xs:annotation>
        <xs:documentation>
            Includes or references a schematron doc for enforcing process constraints
```

```
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="sml:ruleLanguageType">
                    <xs:sequence minOccurs="0">
                        <xs:any namespace="http://www.ascc.net/xml/schematron" processContents="lax"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="relaxNG" substitutionGroup="sml:ruleLanguage">
        <xs:annotation>
            <xs:documentation>
                Includes or references a relaxNG doc for enforcing process constraints
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="sml:ruleLanguageType">
                    <xs:sequence minOccurs="0">
                        <xs:any namespace="http://relaxng.org/ns/structure/1.0" processContents="lax"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

A.4 system.xsd.

The system.xsd schema defines a System and Component derived from Process, with additional positional and interface information.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/sensorML/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--  -->
    <xs:annotation>
        <xs:documentation>Component and System objects for core SensorML</xs:documentation>
    </xs:annotation>
    <!-- ================================================= -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:import namespace="http://www.opengis.net/swe/1.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/1.0.0/swe.xsd"/>
    <xs:include schemaLocation="./process.xsd"/>
    <!-- ================================================= -->
    <xs:complexType name="AbstractDerivableComponentType" abstract="true">
        <xs:annotation>
            <xs:documentation>
                Complex Type to allow creation of component profiles by extension
            </xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="sml:AbstractProcessType">
                <xs:sequence>
                    <xs:element ref="sml:spatialReferenceFrame" minOccurs="0"/>
                    <xs:element ref="sml:temporalReferenceFrame" minOccurs="0"/>
```

```
                <xs:choice minOccurs="0">
                    <xs:element ref="sml:location"/>
                    <xs:element ref="sml:position"/>
                </xs:choice>
                <xs:element ref="sml:timePosition" minOccurs="0"/>
                <xs:element ref="sml:interfaces" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="AbstractComponentType" abstract="true">
    <xs:annotation>
        <xs:documentation>
            Complex Type for all generic components (soft typed inputs/outputs/parameters)
        </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="sml:AbstractDerivableComponentType">
            <xs:sequence>
                <xs:element ref="sml:inputs" minOccurs="0"/>
                <xs:element ref="sml:outputs" minOccurs="0"/>
                <xs:element ref="sml:parameters" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="spatialReferenceFrame">
    <xs:annotation>
        <xs:documentation>
            Textual definition of a spatial frame axes (origin, orientation). Spatial frames can be related to one
            another by specifying relative positions.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="gml:EngineeringCRS"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="temporalReferenceFrame">
    <xs:annotation>
        <xs:documentation>
            Textual definition of a temporal frame (time origin). Temporal frames can be related to one another
            by specifying relative times.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="gml:TemporalCRS"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<!-- ================================================ -->
<xs:element name="location">
    <xs:annotation>
        <xs:documentation>
            Uses a gml:Point for a fixed location or a (time dependant) curve for time variable location
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:choice minOccurs="0">
            <xs:element ref="gml:Point"/>
            <xs:element ref="gml:_Curve"/>
        </xs:choice>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
```

```
    </xs:complexType>
</xs:element>
<!-- ================================================= -->
<xs:element name="position">
    <xs:annotation>
        <xs:documentation>
            Full position (location + orientation) given by a swe:Position or a Process (if variable)
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:choice minOccurs="0">
            <xs:element ref="sml:_Process"/>
            <xs:element ref="swe:Position"/>
            <xs:element ref="swe:Vector"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:token" use="required"/>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================= -->
<xs:element name="timePosition">
    <xs:annotation>
        <xs:documentation>
            Provide the ability to relate  a local time frame to a reference time frame
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:choice minOccurs="0">
            <xs:element ref="swe:Time"/>
            <xs:element ref="sml:_Process"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:token" use="required"/>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================= -->
<xs:element name="Component" type="sml:ComponentType" substitutionGroup="sml:_Process">
    <xs:annotation>
        <xs:documentation>Atomic SensorML Component</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="ComponentType">
    <xs:complexContent>
        <xs:extension base="sml:AbstractComponentType">
            <xs:sequence>
                <xs:element ref="sml:method" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================= -->
<xs:element name="System" type="sml:SystemType" substitutionGroup="sml:_Process">
    <xs:annotation>
        <xs:documentation>
            System is a composite component containing sub-components.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="SystemType">
    <xs:complexContent>
        <xs:extension base="sml:AbstractComponentType">
            <xs:sequence>
                <xs:element ref="sml:components" minOccurs="0"/>
                <xs:element ref="sml:positions" minOccurs="0"/>
                <xs:element ref="sml:connections" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
```

```
    </xs:complexContent>
</xs:complexType>
<!-- ================================================== -->
<xs:element name="positions">
    <xs:annotation>
        <xs:documentation>Relative positions of the System components</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="PositionList">
                <xs:complexType>
                    <xs:choice>
                        <xs:element ref="sml:position" maxOccurs="unbounded"/>
                        <xs:element ref="sml:timePosition"/>
                    </xs:choice>
                    <xs:attribute name="id" type="xs:ID" use="optional"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="interfaces">
    <xs:annotation>
        <xs:documentation>
            List of interfaces useable to access System inputs and outputs
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element name="InterfaceList">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="sml:interface" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:ID" use="optional"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="interface">
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element ref="sml:InterfaceDefinition"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:token" use="required"/>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:element name="InterfaceDefinition">
    <xs:annotation>
        <xs:documentation>
            Interface definition based on the OSI model. (http://en.wikipedia.org/wiki/OSI_model)
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="serviceLayer" type="sml:LayerPropertyType" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>
                        Layer 8 (not in OSI). Type of web service used to access the data. (Ex: SOS, WCS,
                        WFS)
```

```
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="applicationLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 7 of the OSI model. Provides a means for the user to access information on the
                network through an application. (Ex: HTTP, SMTP, FTP, XMPP, Telnet, NTP, RTP,
                NFS)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="presentationLayer" type="sml:PresentationLayerPropertyType"
        minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 6 of the OSI model. Transforms the data to provide a standard interface for the
                Application layer. (Ex: SSL, TLS, ASCII, MIDI, MPEG, SWECommon)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="sessionLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 5 of the OSI model. Controls the dialogues (sessions) between computers by
                establishing, managing and terminating connections between the local and remote
                application. (Ex: NetBios, TCP session establishment)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="transportLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 4 of the OSI model. Provides transparent transfer of data between end users
                and can control reliability of a given link. (Ex: TCP, UDP, SPX)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="networkLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 3 of the OSI model. Provides functional and procedural means of transfering
                data from source to destination via one or more networks while insuring QoS. (Ex: IP,
                ICMP, ARP, IPSec, IPX)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="dataLinkLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 2 of the OSI model. Provides functional and procedural means of transfering
                data between network entities and detecting/correcting errors. (Ex: Ethernet, 802.11,
                Token ring, ATM, Fibre Channel)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="physicalLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
            <xs:documentation>
                Layer 1 of the OSI model. Provides all electrical and physical characteristics of the
                connection including pin layouts, voltages, cables specifcations, etc... (Ex: RS232,
                100BASE-T, DSL, 802.11g)
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="mechanicalLayer" type="sml:LayerPropertyType" minOccurs="0">
        <xs:annotation>
```

```
                        <xs:documentation>
                            Layer 0 (not is OSI). Type of connector used. (Ex: DB9, DB25, RJ45, RJ11,
                            MINIDIN-8, USB-A, USB-B)
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="optional"/>
        </xs:complexType>
    </xs:element>
    <!-- ================================================ -->
    <xs:complexType name="LayerPropertyType">
        <xs:choice minOccurs="0">
            <xs:element ref="swe:AbstractDataRecord"/>
            <xs:element ref="swe:Category"/>
        </xs:choice>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:complexType name="PresentationLayerPropertyType">
        <xs:choice minOccurs="0">
            <xs:element ref="swe:AbstractDataRecord"/>
            <xs:element ref="swe:Category"/>
            <xs:element ref="swe:DataBlockDefinition"/>
            <xs:element ref="swe:DataStreamDefinition"/>
        </xs:choice>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:element name="ComponentArray" type="sml:ComponentArrayType" substitutionGroup="sml:_Process">
        <xs:annotation>
            <xs:documentation>
                Special Type of system used to describe large arrays of almost identical components. An indexing
                 mechanism can be used to vary certain parameters according to one or more indices value
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ================================================ -->
    <xs:complexType name="ComponentArrayType">
        <xs:complexContent>
            <xs:extension base="sml:AbstractDerivableComponentType">
                <xs:sequence>
                    <xs:element ref="sml:inputs" minOccurs="0"/>
                    <xs:element ref="sml:outputs" minOccurs="0"/>
                    <xs:element name="parameters">
                        <xs:complexType>
                            <xs:complexContent>
                                <xs:restriction base="sml:parametersPropertyType">
                                    <xs:sequence minOccurs="0">
                                        <xs:element name="ParameterList">
                                            <xs:complexType>
                                                <xs:complexContent>
                                                    <xs:restriction base="sml:AbstractListType">
                                                        <xs:sequence>
                                                            <xs:element name="index" maxOccurs="unbounded">
                                                                <xs:complexType>
                                                                    <xs:sequence>
                                                                        <xs:element ref="swe:Count"/>
                                                                    </xs:sequence>
                                                                    <xs:attribute name="name" type="xs:token"/>
                                                                </xs:complexType>
                                                            </xs:element>
                                                            <xs:element name="parameter"
                                                                type="swe:DataComponentPropertyType"
                                                                maxOccurs="unbounded"/>
                                                        </xs:sequence>
                                                    </xs:restriction>
```

```
                                        </xs:complexContent>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:restriction>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element ref="sml:components" minOccurs="0"/>
            <xs:element ref="sml:positions" minOccurs="0"/>
            <xs:element ref="sml:connections" minOccurs="0"/>
        </xs:sequence>
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:schema>
```

A.5 sensorML.xsd.

The SensorML element serves as a document container for SensorML components.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/sensorML/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:sml="http://www.opengis.net/sensorML/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--  -->
    <xs:annotation>
        <xs:documentation>SensorML document root definition</xs:documentation>
    </xs:annotation>
    <!-- ================================================= -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:import namespace="http://www.opengis.net/swe/1.0"
        schemaLocation="http://schemas.opengis.net/sweCommon/1.0.0/swe.xsd"/>
    <xs:include schemaLocation="./system.xsd"/>
    <!-- ================================================= -->
    <xs:element name="SensorML">
        <xs:annotation>
            <xs:documentation>SensorML document root</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:group ref="sml:metadataGroup"/>
                <xs:element name="member" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:choice>
                            <xs:element ref="sml:_Process"/>
                            <xs:element ref="sml:DocumentList"/>
                            <xs:element ref="sml:ContactList"/>
                        </xs:choice>
                        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="version" type="xs:token" use="required" fixed="1.0"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Annex B. XML Schemas for SWE Common (normative)

B.1 swe.xsd

Stub schema for importing in all of swe Common.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <xs:annotation>
        <xs:documentation>Stub schema for swe</xs:documentation>
    </xs:annotation>
    <!--========================-->
    <!--   <xs:include schemaLocation="./basicTypes.xsd"/>  included by simpleTypes.xsd -->
    <!--   <xs:include schemaLocation="./simpleTypes.xsd"/> included by aggregateTypes.xsd -->
    <!--   <xs:include schemaLocation="./aggregateTypes.xsd"/> included by positionTypes, curveTypes & data -->
    <xs:include schemaLocation="./positionTypes.xsd"/>
    <xs:include schemaLocation="./curveTypes.xsd"/>
    <xs:include schemaLocation="./data.xsd"/>
    <xs:include schemaLocation="./temporalAggregates.xsd"/>
    <xs:include schemaLocation="./phenomenon.xsd"/>
    <xs:include schemaLocation="./xmlData.xsd"/>
    <!--========================-->
</xs:schema>
```

B.2 basicTypes.xsd.

Some basic types (simpleContent) required in various places in Sensor Web Enablement application schemas.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
    <xs:annotation>
        <xs:documentation>basicTypes.xsd
            Some basic types (simpleContent) required in various places in OWS Sensor Web application
            schemas;

            Copyright (c) 2007 Open Geospatial Consortium - see http://www.opengeospatial.org/about/?page=ipr
        </xs:documentation>
    </xs:annotation>
    <!-- ============================================================================ -->
    <xs:import namespace="http://www.w3.org/1999/xlink"
        schemaLocation="http://schemas.opengis.net/xlink/1.0.0/xlinks.xsd"/>
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <!-- ============================================================================ -->
    <!-- === Units of Measure === -->
    <!-- ============================================================================ -->
    <xs:simpleType name="UomSymbol">
        <xs:annotation>
            <xs:documentation>
                Local copy of GML 3.2 uom symbol definition included for forward compatibility.
```

```
                Note: in future of this specification based on GML 3.2, these will be removed in favour of the GML
                 3.2 implementation
            </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:pattern value="[^: \n\r\t]+"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- ..... -->
    <xs:simpleType name="UomURI">
        <xs:annotation>
            <xs:documentation>Local copy of GML 3.2 uom URI definition</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:anyURI">
            <xs:pattern value="([a-zA-Z][a-zA-Z0-9\-\+\.]*:|\.\./|\./|#).*"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- ..... -->
    <xs:simpleType name="UomIdentifier">
        <xs:annotation>
            <xs:documentation>Local copy of GML 3.2 uom identifier definition</xs:documentation>
        </xs:annotation>
        <xs:union memberTypes="swe:UomSymbol swe:UomURI"/>
    </xs:simpleType>
    <!-- ..... -->
    <xs:complexType name="UomPropertyType">
        <xs:annotation>
            <xs:documentation>
                Property type that indicates unit-of-measure, either by (i) inline definition; (ii)  reference; (iii)  UCUM
                code
            </xs:documentation>
        </xs:annotation>
        <xs:sequence minOccurs="0">
            <xs:element ref="gml:UnitDefinition">
                <xs:annotation>
                    <xs:documentation>Defines a unit inline</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="code" type="swe:UomSymbol">
            <xs:annotation>
                <xs:documentation>Specifies a unit by using a UCUM expression (prefered)</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup">
            <xs:annotation>
                <xs:documentation>
                    Specifies a unit by referencing a dictionary entry (to be used only for units not defined in
                    UCUM)
                </xs:documentation>
            </xs:annotation>
        </xs:attributeGroup>
    </xs:complexType>
    <!-- ========================================================================= -->
    <!-- === Code space/vocabulary === -->
    <!-- ========================================================================= -->
    <xs:complexType name="CodeSpacePropertyType">
        <xs:annotation>
            <xs:documentation>Property type that indicates the codespace</xs:documentation>
        </xs:annotation>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ========================================================================= -->
    <!-- === Time position === -->
    <!-- overlaps with GML TimePositionType -->
    <!-- ========================================================================= -->
    <xs:simpleType name="timeIso8601">
```

```
        <xs:annotation>
            <xs:documentation>
                Choice of time position encodings, not including numeric representation.
                A minor variation on gml:TimePositionUnion - carrying "indeterminate value" as content instead of
                an attribute.
            </xs:documentation>
        </xs:annotation>
        <xs:union memberTypes="xs:date xs:time xs:dateTime gml:TimeIndeterminateValueType"/>
    </xs:simpleType>
    <!-- ..... -->
    <xs:simpleType name="timePositionType">
        <xs:annotation>
            <xs:documentation>
                Choice of time position encodings, including numeric representation but no frame.
                A minor variation on gml:TimePositionUnion - carrying "indeterminate value" as content instead of
                an attribute.
            </xs:documentation>
        </xs:annotation>
        <xs:union memberTypes="swe:timeIso8601 xs:double"/>
    </xs:simpleType>
    <!-- ========================================================================== -->
    <!-- === Scoped Name === -->
    <!-- equivalent to GML 3.2 CodeWithAuthorityType -->
    <!-- ========================================================================== -->
    <xs:complexType name="ScopedNameType">
        <xs:annotation>
            <xs:documentation>
                Explicit implementation of ISO 19103 ScopedName.
                Extension of string which also carries a codeSpace attribute.
                Note: in future versions of this specification based on GML 3.2, this will be removed in favour of
                gml:CodeWithAuthorityType.
            </xs:documentation>
        </xs:annotation>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="codeSpace" type="xs:anyURI" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <!-- ========================================================================== -->
    <!-- === Lists === -->
    <!-- ================================================ -->
    <xs:simpleType name="decimalList">
        <xs:annotation>
            <xs:documentation>
                Simple list of double-precision numbers.
                Note: xs:double supports either decimal or scientific notation
            </xs:documentation>
        </xs:annotation>
        <xs:list itemType="xs:double"/>
    </xs:simpleType>
    <!-- ..... -->
    <xs:simpleType name="decimalPair">
        <xs:annotation>
            <xs:documentation>
                Pair of double-precision numbers.
                Note: xs:double supports either decimal or scientific notation
            </xs:documentation>
        </xs:annotation>
        <xs:restriction base="swe:decimalList">
            <xs:length value="2"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- ================================================ -->
    <xs:simpleType name="countList">
        <xs:annotation>
            <xs:documentation>Simple list of integers. </xs:documentation>
```

```
    </xs:annotation>
    <xs:list itemType="xs:integer"/>
</xs:simpleType>
<!-- ..... -->

    <xs:annotation>
        <xs:documentation>Pair of integers. </xs:documentation>
    </xs:annotation>

        <xs:length value="2"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tokenList">
    <xs:annotation>
        <xs:documentation>

            Note: xs:token is a string with no embedded white-space allowed
        </xs:documentation>
    </xs:annotation>

</xs:simpleType>
<!-- ================================================ -->
<xs:simpleType name="timeList">
    <xs:annotation>
        <xs:documentation>Simple list of time positions. </xs:documentation>
    </xs:annotation>
    <xs:list itemType="swe:timePositionType"/>
</xs:simpleType>
<!-- ..... -->
<xs:simpleType name="timePair">
    <xs:annotation>

    </xs:annotation>
    <xs:restriction base="swe:timeList">

    </xs:restriction>
</xs:simpleType>

<!-- ===== Soft-typed values ====== -->
<!-- ================================================================== -->
<xs:complexType name="TypedValueType">
    <xs:annotation>
        <xs:documentation>A generic soft-typed value</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="property" type="swe:ScopedNameType">
            <xs:annotation>
                <xs:documentation>
                    This element attribute indicates the semantics of the typed value.
                    Usually identifies a property or phenomenon definition.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="value">
            <xs:annotation>
                <xs:documentation>Implicit xs:anyType</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<!-- ..... -->
<xs:element name="TypedValue" type="swe:TypedValueType">
    <xs:annotation>
        <xs:documentation>A generic soft-typed value</xs:documentation>
    </xs:annotation>
</xs:element>
```

```
<!-- ..... -->
<xs:complexType name="TypedValuePropertyType">
    <xs:annotation>
        <xs:documentation>Inline property type for soft-typed values</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="swe:TypedValue"/>
    </xs:sequence>
</xs:complexType>
<!-- ========================================================================== -->
<xs:complexType name="TypedValueListType">
    <xs:annotation>
        <xs:documentation>A list of generic soft-typed values</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="property" type="swe:ScopedNameType">
            <xs:annotation>
                <xs:documentation>
                    This element attribute indicates the semantics of the typed value.
                    Usually identifies a property or phenomenon definition.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="value" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>Implicit xs:anyType</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<!-- ..... -->
<xs:element name="TypedValueList" type="swe:TypedValueListType">
    <xs:annotation>
        <xs:documentation>A generic soft-typed list of values</xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ..... -->
<xs:complexType name="TypedValueListPropertyType">
    <xs:annotation>
        <xs:documentation>Inline property type for list of generic soft-typed values</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="swe:TypedValueList"/>
    </xs:sequence>
</xs:complexType>
<!-- ========================================================================== -->
<!-- === Generic Interval === -->
<!-- ========================================================================== -->
<xs:complexType name="IntervalType">
    <xs:annotation>
        <xs:documentation>
            A generic interval. The type of the two limits will normally be the same.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="lowerBound">
            <xs:annotation>
                <xs:documentation>Implicit xs:anyType</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="upperBound">
            <xs:annotation>
                <xs:documentation>Implicit xs:anyType</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

```
    <!-- ..... -->
    <xs:element name="Interval" type="swe:IntervalType">
        <xs:annotation>
            <xs:documentation>
                A generic interval. The type of the two limits will normally be the same.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ..... -->
    <xs:complexType name="IntervalPropertyType">
        <xs:annotation>
            <xs:documentation>Inline property type for generic intervals</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:Interval"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ========================================================================== -->
</xs:schema>
```

B.3 simpleTypes.xsd.

The simpleTypes.xsd schema defines simple data types used throughout SensorML and the Sensor Web Enablement framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
    <xs:annotation>
        <xs:documentation>
            SWE common schema for simple data types (i.e. without children)

            These have mostly been implemented using the "composition" pattern, rather than using
                inheritance/type-derivation.
            Note: In general, content model definition using composition is more easily accomplished using groups
                and attribute groups.
            * global/named type definitions are not needed unless type derivation is required.
            * type derivation is not needed unless substitution groups are to be supported.
            * parent types that are too generalized will require derivation by restriction further down, so should be
                avoided.

            Copyright (c) 2007 Open Geospatial Consortium - see http://www.opengeospatial.org/about/?page=ipr
        </xs:documentation>
    </xs:annotation>
    <!-- ================================================ -->
    <xs:import namespace="http://www.w3.org/1999/xlink"
        schemaLocation="http://schemas.opengis.net/xlink/1.0.0/xlinks.xsd"/>
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:include schemaLocation="./basicTypes.xsd"/>
    <!-- ================================================ -->
    <!-- === Root class model === -->
    <!-- ================================================ -->
    <xs:complexType name="AbstractDataComponentType" abstract="true">
        <xs:annotation>
            <xs:documentation>
                Base type for all data components.
                This is implemented as an XML Schema complexType because it includes both element and
                attribute content.
```

```
            </xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="gml:AbstractGMLType">
                <xs:attribute name="fixed" type="xs:boolean" use="optional" default="false">
                    <xs:annotation>
                        <xs:documentation>
                            Specifies if the value of a component stays fixed in time or is variable. Default is
                            variable
                        </xs:documentation>
                    </xs:annotation>
                </xs:attribute>
                <xs:attribute name="definition" type="xs:anyURI" use="optional">
                    <xs:annotation>
                        <xs:documentation>
                            Points to semantics information defining the precise nature of the component
                        </xs:documentation>
                    </xs:annotation>
                </xs:attribute>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================= -->
    <!--              Simple Components                    -->
    <!-- ================================================= -->
    <xs:attributeGroup name="SimpleComponentAttributeGroup">
        <xs:annotation>
            <xs:documentation>
                Basic attributes required for all simple data components (i.e. without children)
            </xs:documentation>
        </xs:annotation>
        <xs:attribute name="referenceFrame" type="xs:anyURI" use="optional">
            <xs:annotation>
                <xs:documentation>
                    A reference frame anchors a value to a datum or interval scale
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="axisID" type="xs:token" use="optional">
            <xs:annotation>
                <xs:documentation>
                    Specifies the reference axis using the gml:axisID. The reference frame URI is inherited from
                    parent Vector
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:attributeGroup>
    <!-- ================================================= -->
    <!-- ======== Booleans ============= -->
    <!-- ================================================= -->
    <xs:element name="Boolean">
        <xs:annotation>
            <xs:documentation>
                Scalar component used to express truth: True or False, 0 or 1
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType">
                    <xs:sequence>
                        <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The quality property provides an indication of the reliability of estimates of the
                                    asociated value
                                </xs:documentation>
                            </xs:annotation>
```

```
                    </xs:element>
                    <xs:element name="value" type="xs:boolean" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                Value is optional, to enable structure to act in a schema for values provided
                                using other encodings
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:sequence>
                <xs:attributeGroup ref="swe:SimpleComponentAttributeGroup"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- ..... -->
<xs:complexType name="BooleanPropertyType">
    <xs:annotation>
        <xs:documentation>
            Boolean is a data-type so usually appears "by value" rather than by reference.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="swe:Boolean"/>
    </xs:sequence>
</xs:complexType>
<!-- ================================================ -->
<!-- ======== Quantities ============= -->
<!-- ================================================ -->
<xs:element name="Quantity">
    <xs:annotation>
        <xs:documentation>Decimal number with optional unit and constraints</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataComponentType">
                <xs:sequence>
                    <xs:element name="uom" type="swe:UomPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>Unit of measure</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="constraint" type="swe:AllowedValuesPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                The constraint property defines the permitted values, as a range or
                                enumerated list
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0"
                        maxOccurs="unbounded">
                        <xs:annotation>
                            <xs:documentation>
                                The quality property provides an indication of the reliability of estimates of the
                                asociated value
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="value" type="xs:double" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                Value is optional, to enable structure to act in a schema for values provided
                                using other encodings
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
```

```
                </xs:sequence>
                <xs:attributeGroup ref="swe:SimpleComponentAttributeGroup"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- ........ -->
<xs:complexType name="QuantityPropertyType">
    <xs:annotation>
        <xs:documentation>
            Quantity is a data-type so usually appears "by value" rather than by reference.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="swe:Quantity"/>
    </xs:sequence>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="QuantityRange">
    <xs:annotation>
        <xs:documentation>Decimal pair for specifying a quantity range with constraints</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataComponentType">
                <xs:sequence>
                    <xs:element name="uom" type="swe:UomPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>Unit of measure</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="constraint" type="swe:AllowedValuesPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                The constraint property defines the permitted values, as a range or
                                enumerated list
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0"
                        maxOccurs="unbounded">
                        <xs:annotation>
                            <xs:documentation>
                                The quality property provides an indication of the reliability of estimates of the
                                asociated value
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="value" type="swe:decimalPair" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                Value is optional, to enable structure to act in a schema for values provided
                                using other encodings
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:sequence>
                <xs:attributeGroup ref="swe:SimpleComponentAttributeGroup"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- ........ -->
<xs:complexType name="QuantityRangePropertyType">
    <xs:annotation>
        <xs:documentation>
            QuantityRange is a data-type so usually appears "by value" rather than by reference.
```

```
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:QuantityRange"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:element name="Count">
        <xs:annotation>
            <xs:documentation>Integer number used for a counting value</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType">
                    <xs:sequence>
                        <xs:element name="constraint" type="swe:AllowedValuesPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The constraint property defines the permitted values, as a range or
                                    enumerated list
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0"
                            maxOccurs="unbounded">
                            <xs:annotation>
                                <xs:documentation>
                                    The quality property provides an indication of the reliability of estimates of the
                                    asociated value
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="value" type="xs:integer" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Value is optional, to enable structure to act in a schema for values provided
                                    using other encodings
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <!--
                        <xs:element name="uom" type="swe:UomPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>Unit of measure</xs:documentation>
                            </xs:annotation>
                        If there is a UOM then it is a scaled Quantity, not a Count.
                        </xs:element>
                        -->
                    </xs:sequence>
                    <xs:attributeGroup ref="swe:SimpleComponentAttributeGroup"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ..... -->
    <xs:complexType name="CountPropertyType">
        <xs:annotation>
            <xs:documentation>
                Count is a data-type so usually appears "by value" rather than by reference.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:Count"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:element name="CountRange">
```

```
        <xs:annotation>
            <xs:documentation>Integer pair used for specifying a count range</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType">
                    <xs:sequence>
                        <xs:element name="constraint" type="swe:AllowedValuesPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The constraint property defines the permitted values, as a range or
                                    enumerated list
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0"
                            maxOccurs="unbounded">
                            <xs:annotation>
                                <xs:documentation>
                                    The quality property provides an indication of the reliability of estimates of the
                                    asociated value
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="value" type="swe:countPair" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Value is optional, to enable structure to act in a schema for values provided
                                    using other encodings
                                </xs:documentation>
                            </xs:annotation>
                            <!-- *** Maybe cleaner to implement as a pair of value elements? *** -->
                        </xs:element>
                        <!--
                        <xs:element name="uom" type="swe:UomPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>Unit of measure</xs:documentation>
                            </xs:annotation>
                        If there is a UOM then it is a scaled Quantity, not a Count.
                        </xs:element>
                        -->
                    </xs:sequence>
                    <xs:attributeGroup ref="swe:SimpleComponentAttributeGroup"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ..... -->
    <xs:complexType name="CountRangePropertyType">
        <xs:annotation>
            <xs:documentation>
                CountRange is a data-type so usually appears "by value" rather than by reference.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:CountRange"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ================================================ -->
    <!-- ======== Categories ============= -->
    <!-- ================================================ -->
    <xs:element name="Category">
        <xs:annotation>
            <xs:documentation>
                A simple token identifying a term or category (single spaces allowed); definition attribute should
                provide dictionary entry useful for interpretation of the value
            </xs:documentation>
```

```
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType">
                    <xs:sequence>
                        <xs:element name="codeSpace" type="swe:CodeSpacePropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Provides link to dictionary or rule set to which the value belongs
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="constraint" type="swe:AllowedTokensPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The constraint property defines the permitted values, as an enumerated list
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The quality property provides an indication of the reliability of estimates of the
                                    asociated value
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="value" type="xs:token" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Value is optional, to enable structure to act in a schema for values provided
                                    using other encodings
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                    <xs:attributeGroup ref="swe:SimpleComponentAttributeGroup"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ........ -->
    <xs:complexType name="CategoryPropertyType">
        <xs:annotation>
            <xs:documentation>
                Category is a data-type so usually appears "by value" rather than by reference.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:Category"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ================================================ -->
    <!-- ========== Time Components ======== -->
    <!-- ================================================ -->
    <xs:attributeGroup name="TRSAttributeGroup">
        <xs:attribute name="referenceTime" type="swe:timeIso8601" use="optional">
            <xs:annotation>
                <xs:documentation>
                    Specifies the origin of the temporal reference frame as an ISO8601 date (used to specify time
                    after an epoch)
                </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="referenceFrame" type="xs:anyURI" use="optional">
            <xs:annotation>
                <xs:documentation>
```

```
                Points to a temporal reference frame definition. Time value will be expressed relative to this
                frame
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="localFrame" type="xs:anyURI" use="optional">
        <xs:annotation>
            <xs:documentation>
                Specifies the temporal frame which origin is given by this time component
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:attributeGroup>
<!-- ================================================ -->
<xs:element name="Time">
    <xs:annotation>
        <xs:documentation>
            Either ISO 8601 (e.g. 2004-04-18T12:03:04.6Z) or time relative to a time origin
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataComponentType">
                <xs:sequence>
                    <xs:element name="uom" type="swe:UomPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>Unit of measure</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="constraint" type="swe:AllowedTimesPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                The constraint property defines the permitted values, as a range or
                                enumerated list
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                The quality property provides an indication of the reliability of estimates of the
                                asociated value
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="value" type="swe:timePositionType" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>
                                Value is optional, to enable structure to act in a schema for values provided
                                using other encodings
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:sequence>
                <xs:attributeGroup ref="swe:TRSAttributeGroup"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- ..... -->
<xs:complexType name="TimePropertyType">
    <xs:annotation>
        <xs:documentation>
            Time is a data-type so usually appears "by value" rather than by reference.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
```

```
            <xs:element ref="swe:Time"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:element name="TimeRange">
        <xs:annotation>
            <xs:documentation>
                Time value pair for specifying a time range (can be a decimal or ISO 8601)
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType">
                    <xs:sequence>
                        <xs:element name="uom" type="swe:UomPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>Unit of measure</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="constraint" type="swe:AllowedTimesPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The constraint property defines the permitted values, as a range or
                                    enumerated list
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="quality" type="swe:QualityPropertyType" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    The quality property provides an indication of the reliability of estimates of the
                                    asociated value
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="value" type="swe:timePair" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Value is optional, to enable structure to act in a schema for values provided
                                    using other encodings
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                    <xs:attributeGroup ref="swe:TRSAttributeGroup"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ..... -->
    <xs:complexType name="TimeRangePropertyType">
        <xs:annotation>
            <xs:documentation>
                TimeRange is a data-type so usually appears "by value" rather than by reference.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:TimeRange"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ================================================ -->
    <!-- ======== Text Object  ============= -->
    <!-- ================================================ -->
    <xs:element name="Text">
        <xs:annotation>
            <xs:documentation>Free textual component</xs:documentation>
        </xs:annotation>
```

```
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType">
                    <xs:sequence>
                        <xs:element name="value" type="xs:string" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>
                                    Value is optional, to enable structure to act in a schema for values provided
                                    using other encodings
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ..... -->
    <xs:complexType name="TextPropertyType">
        <xs:annotation>
            <xs:documentation>
                Text is a data-type so usually appears "by value" rather than by reference.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:Text"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ================================================ -->
    <!-- ======== ObservableProperty ============= -->
    <!-- ================================================ -->
    <xs:element name="ObservableProperty">
        <xs:annotation>
            <xs:documentation>
                ObservableProperty should be used to identify (through reference only) stimuli or measurable
                property types. The consequence is that it does not have a uom because it has not been
                measured yet.  This is used to define sensor/detector/actuator inputs and outputs, for instance.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractDataComponentType"/>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ================================================ -->
    <!-- ============ Constraints and Qualifiers ================ -->
    <!-- ================================================ -->
    <xs:complexType name="QualityPropertyType">
        <xs:annotation>
            <xs:documentation>
                Allows for a simple quality assessment of the values carried by this component.
                This value can be numerical or categorical thus allowing for things like accuracy, precision,
                tolerance, confidence level, etc...

                The  meaning of the quality measure is indicated by the definition attribute of the chosen sub-
                component.

                The use of the 'ref'attribute indicate that the value of accuracy is included itself in the data inside
                the referred component.

                This soft-typed Data Quality description may be replaced by ISO 19115/19139 DQ_DataQuality
                elements in later versions
            </xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="0">
            <xs:element ref="swe:Quantity"/>
```

```
            <xs:element ref="swe:QuantityRange"/>
            <xs:element ref="swe:Category"/>
            <xs:element ref="swe:Text"/>
        </xs:choice>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ============================================== -->
    <xs:complexType name="AllowedValuesPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element ref="swe:AllowedValues"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ........ -->
    <xs:element name="AllowedValues">
        <xs:annotation>
            <xs:documentation>
                List of allowed values (There is an implicit AND between all members)
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice>
                <xs:choice>
                    <xs:element name="min" type="xs:double">
                        <xs:annotation>
                            <xs:documentation>
                                Specifies minimum allowed value for an open interval (no max)
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="max" type="xs:double">
                        <xs:annotation>
                            <xs:documentation>
                                Specifies maximum allowed value for an open interval (no min)
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:choice>
                <xs:choice maxOccurs="unbounded">
                    <xs:element name="interval" type="swe:decimalPair">
                        <xs:annotation>
                            <xs:documentation>
                                Range of allowed values (closed interval) for this component
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="valueList" type="swe:decimalList">
                        <xs:annotation>
                            <xs:documentation>List of allowed values for this component</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:choice>
            </xs:choice>
            <xs:attribute name="id" type="xs:ID"/>
        </xs:complexType>
    </xs:element>
    <!-- ============================================== -->
    <xs:complexType name="AllowedTokensPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element ref="swe:AllowedTokens"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ........ -->
    <xs:element name="AllowedTokens">
        <xs:annotation>
            <xs:documentation>Enumeration of allowed values</xs:documentation>
```

```
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="valueList" type="swe:tokenList" maxOccurs="unbounded">
                    <xs:annotation>
                        <xs:documentation>List of allowed token values for this component</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID"/>
        </xs:complexType>
    </xs:element>
    <!-- ============================================== -->
    <xs:complexType name="AllowedTimesPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element ref="swe:AllowedTimes"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ........ -->
    <xs:element name="AllowedTimes">
        <xs:annotation>
            <xs:documentation>
                List of allowed time values (There is an implicit AND between all members)
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice>
                <xs:choice>
                    <xs:element name="min" type="swe:timePositionType">
                        <xs:annotation>
                            <xs:documentation>
                                Specifies minimum allowed time value for an open interval (no max)
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="max" type="swe:timePositionType">
                        <xs:annotation>
                            <xs:documentation>
                                Specifies maximum allowed time value for an open interval (no min)
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:choice>
                <xs:choice maxOccurs="unbounded">
                    <xs:element name="interval" type="swe:timePair">
                        <xs:annotation>
                            <xs:documentation>
                                Range of allowed time values (closed interval) for this component
                            </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="valueList" type="swe:timeList">
                        <xs:annotation>
                            <xs:documentation>List of allowed time values for this component</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:choice>
            </xs:choice>
            <xs:attribute name="id" type="xs:ID"/>
        </xs:complexType>
    </xs:element>
    <!-- ============================================== -->
    <!--            Data Component Groups               -->
    <!-- ============================================== -->
    <xs:group name="AnyScalar">
        <xs:annotation>
```

```
            <xs:documentation>Re-usable group providing a choice of scalar data types</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:group ref="swe:AnyNumerical"/>
            <xs:element ref="swe:Boolean"/>
            <xs:element ref="swe:Category"/>
            <xs:element ref="swe:Text"/>
        </xs:choice>
    </xs:group>
    <!-- ........ -->
    <xs:complexType name="AnyScalarPropertyType">
        <xs:annotation>
            <xs:documentation>Complex Type for all properties taking the AnyScalar Group</xs:documentation>
        </xs:annotation>
        <xs:group ref="swe:AnyScalar" minOccurs="0"/>
        <xs:attribute name="name" type="xs:token" use="required"/>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:group name="AnyNumerical">
        <xs:annotation>
            <xs:documentation>Re-usable group providing a choice of numeric data types</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element ref="swe:Count"/>
            <xs:element ref="swe:Quantity"/>
            <xs:element ref="swe:Time"/>
        </xs:choice>
    </xs:group>
    <!-- ........ -->
    <xs:group name="AnyRange">
        <xs:annotation>
            <xs:documentation>Re-usable group providing a choice of range data types</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element ref="swe:QuantityRange"/>
            <xs:element ref="swe:CountRange"/>
            <xs:element ref="swe:TimeRange"/>
        </xs:choice>
    </xs:group>
</xs:schema>
```

B.4 aggregateTypes.xsd.

Defines the aggregate data types used throughout SWE.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <xs:annotation>
        <xs:documentation>SWE Common schema for aggregate data types</xs:documentation>
    </xs:annotation>
    <!-- ================================================ -->
    <xs:import namespace="http://www.w3.org/1999/xlink"
        schemaLocation="http://schemas.opengis.net/xlink/1.0.0/xlinks.xsd"/>
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:include schemaLocation="./simpleTypes.xsd"/>
    <xs:include schemaLocation="./encoding.xsd"/>
```

```
<!-- ================================================= -->
<!--        Composite/Aggregate Components - Records and arrays          -->
<!-- ================================================= -->
<xs:element name="AbstractDataRecord" type="swe:AbstractDataRecordType" abstract="true"/>
<!-- ........ -->
<xs:complexType name="AbstractDataRecordType" abstract="true">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataComponentType"/>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================= -->
<xs:element name="DataRecord" type="swe:DataRecordType" substitutionGroup="swe:AbstractDataRecord">
    <xs:annotation>
        <xs:documentation>
            Implementation of ISO-11404 Record datatype. This allows grouping of data components which
            can themselves be Records, Arrays or Simple Types
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="DataRecordType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataRecordType">
            <xs:sequence>
                <xs:element name="field" type="swe:DataComponentPropertyType" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ........ -->
<xs:complexType name="DataRecordPropertyType">
    <xs:annotation>
        <xs:documentation>
            DataRecord is a data-type so usually appears "by value" rather than by reference.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="swe:DataRecord"/>
    </xs:sequence>
</xs:complexType>
<!-- ================================================= -->
<xs:element name="SimpleDataRecord" type="swe:SimpleDataRecordType"
    substitutionGroup="swe:AbstractDataRecord">
    <xs:annotation>
        <xs:documentation>
            Implementation of ISO-11404 Record datatype that takes only simple scalars (i.e. no data
            aggregates)
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="SimpleDataRecordType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataRecordType">
            <xs:sequence>
                <xs:element name="field" type="swe:AnyScalarPropertyType" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ........ -->
<xs:complexType name="SimpleDataRecordPropertyType">
    <xs:annotation>
        <xs:documentation>
            SimpleDataRecord is a data-type so usually appears "by value" rather than by reference.
```

```
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:SimpleDataRecord"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:element name="AbstractDataArray" type="swe:AbstractDataArrayType" abstract="true">
        <xs:annotation>
            <xs:documentation>
                Implemetation of ISO-11404 Array datatype. This defines an array of identical data components
                with a elementCount. Values are given as a block and can be encoded in different ways
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ........ -->
    <xs:complexType name="AbstractDataArrayType" abstract="true">
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataComponentType">
                <xs:sequence>
                    <xs:element name="elementCount">
                        <xs:annotation>
                            <xs:documentation>
                                Specifies the size of the array (i.e. the number of elements of the defined type it
                                contains)
                            </xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:sequence minOccurs="0">
                                <xs:element ref="swe:Count">
                                </xs:element>
                            </xs:sequence>
                            <xs:attribute name="ref" type="xs:IDREF">
                                <xs:annotation>
                                    <xs:documentation>
                                        If present, the array size is variable and should be obtained from the
                                        referenced component.
                                        The referenced component must occur before the array values in a data
                                        stream to allow parsing.
                                    </xs:documentation>
                                </xs:annotation>
                            </xs:attribute>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:element name="DataArray" type="swe:DataArrayType" substitutionGroup="swe:AbstractDataArray">
        <xs:annotation>
            <xs:documentation>
                Implemetation of ISO-11404 Array datatype. This defines an array of identical data components
                with a elementCount. Values are given as a block and can be encoded in different ways
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ........ -->
    <xs:complexType name="DataArrayType">
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataArrayType">
                <xs:sequence>
                    <xs:element name="elementType" type="swe:DataComponentPropertyType"/>
                    <xs:group ref="swe:EncodedValuesGroup" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
```

```
</xs:complexType>
<!-- ........ -->
<xs:complexType name="DataArrayPropertyType">
    <xs:annotation>
        <xs:documentation>
            DataArray is a data-type so usually appears "by value" rather than by reference.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="swe:DataArray"/>
    </xs:sequence>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="DataValuePropertyType">
    <xs:annotation>
        <xs:documentation>Use to point or include data values inline</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="xs:anyType">
            <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<!--        Conditionals            -->
<!-- ================================================ -->
<xs:element name="ConditionalData" type="swe:ConditionalDataType"
    substitutionGroup="swe:AbstractDataRecord">
    <xs:annotation>
        <xs:documentation>List of Conditional Values for a property</xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="ConditionalDataType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataRecordType">
            <xs:sequence>
                <xs:element name="case" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="swe:ConditionalValue" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:token" use="required"/>
                        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="AbstractConditionalType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataRecordType">
            <xs:sequence>
                <xs:element name="condition" maxOccurs="unbounded">
                    <xs:annotation>
                        <xs:documentation>
                            Specifies one or more conditions for which the given value is valid
                        </xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                        <xs:group ref="swe:AnyData" minOccurs="0"/>
                        <xs:attribute name="name" type="xs:token" use="required"/>
                        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                    </xs:complexType>
                </xs:element>
```

```
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="ConditionalValue" type="swe:ConditionalValueType"
    substitutionGroup="swe:AbstractDataRecord">
    <xs:annotation>
        <xs:documentation>Qualifies data (scalar or not) with one or more conditions</xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="ConditionalValueType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractConditionalType">
            <xs:sequence>
                <xs:element name="data">
                    <xs:complexType>
                        <xs:group ref="swe:AnyData" minOccurs="0"/>
                        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<!--            Data Component Groups              -->
<!-- ================================================ -->
<xs:group name="AnyData">
    <xs:choice>
        <xs:group ref="swe:AnyScalar"/>
        <xs:group ref="swe:AnyRange"/>
        <xs:element ref="swe:AbstractDataRecord"/>
        <xs:element ref="swe:AbstractDataArray"/>
    </xs:choice>
</xs:group>
<!-- ........ -->
<xs:complexType name="DataComponentPropertyType">
    <xs:annotation>
        <xs:documentation>Complex Type for all properties taking the AnyData Group</xs:documentation>
    </xs:annotation>
    <xs:group ref="swe:AnyData" minOccurs="0"/>
    <xs:attribute name="name" type="xs:token" use="required"/>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ........ -->
<xs:complexType name="SimpleDataPropertyType">
    <xs:annotation>
        <xs:documentation>Supports a DataGroup/DataArray with inline data values</xs:documentation>
    </xs:annotation>
    <xs:group ref="swe:AnyData" minOccurs="0"/>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ================================================ -->
<xs:group name="EncodedValuesGroup">
    <xs:sequence>
        <xs:element name="encoding" type="swe:BlockEncodingPropertyType">
            <xs:annotation>
                <xs:documentation>
                    Specifies an encoding for the data structure defined by the enclosing element
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="values" type="swe:DataValuePropertyType">
            <xs:annotation>
                <xs:documentation>
```

```
                        Carries the block of values encoded as specified by the encoding element
                    </xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:group>
</xs:schema>
```

B.5 data.xsd.

The data.xsd schema defines data block definitions used in SensorML and throughout the Sensor Web Enablement initiative.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <xs:annotation>
        <xs:documentation>
            Schema allowing definition of structure and encoding of sensor data. Multiplexed streams can also be
            described
        </xs:documentation>
    </xs:annotation>
    <!-- ================================================ -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:include schemaLocation="./aggregateTypes.xsd"/>
    <!-- ================================================ -->
    <xs:element name="DataBlockDefinition" type="swe:DataBlockDefinitionType"/>
    <!-- ........ -->
    <xs:complexType name="DataBlockDefinitionType">
        <xs:sequence>
            <xs:element name="components" type="swe:DataComponentPropertyType"/>
            <xs:element name="encoding" type="swe:BlockEncodingPropertyType"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
    </xs:complexType>
    <!-- ........ -->
    <xs:complexType name="DataBlockDefinitionPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element ref="swe:DataBlockDefinition"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:element name="DataStreamDefinition" type="swe:DataStreamDefinitionType"/>
    <!-- ........ -->
    <xs:complexType name="DataStreamDefinitionType">
        <xs:sequence>
            <xs:element name="streamComponent" type="swe:DataBlockDefinitionPropertyType"
                maxOccurs="unbounded"/>
            <xs:element name="streamEncoding" type="swe:MultiplexedStreamFormatPropertyType"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
    </xs:complexType>
    <!-- ........ -->
    <xs:complexType name="DataStreamDefinitionPropertyType">
        <xs:sequence minOccurs="0">
            <xs:element ref="swe:DataStreamDefinition"/>
```

```
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================== -->
</xs:schema>
```

B.6 encoding.xsd.

Provides encoding definitions for aggregate data.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <xs:annotation>
        <xs:documentation>
            Defines basic ResponseType definition and commonly used sensor characteristics
        </xs:documentation>
    </xs:annotation>
    <!-- ================================================== -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <!-- ================================================== -->
    <xs:complexType name="BlockEncodingPropertyType">
        <xs:choice minOccurs="0">
            <xs:element ref="swe:StandardFormat"/>
            <xs:element ref="swe:BinaryBlock"/>
            <xs:element ref="swe:TextBlock"/>
            <xs:element ref="swe:XMLBlock"/>
        </xs:choice>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================== -->
    <xs:complexType name="AbstractEncodingType">
        <xs:attribute name="id" type="xs:ID"/>
        <!-- why bother with the supertype? Not used to form a substitution group. -->
        <!--->Remember - attributes are Optional by default -->
    </xs:complexType>
    <!-- ================================================== -->
    <!-- ================================================== -->
    <xs:element name="StandardFormat">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractEncodingType">
                    <xs:attribute name="mimeType" type="xs:token" use="required"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- ================================================== -->
    <!-- ================================================== -->
    <xs:element name="TextBlock">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="swe:AbstractEncodingType">
                    <xs:attribute name="tokenSeparator" type="swe:textSeparator" use="required"/>
                    <xs:attribute name="blockSeparator" type="swe:textSeparator" use="required"/>
                    <xs:attribute name="decimalSeparator" type="swe:decimalSeparator" use="required"/>
                </xs:extension>
            </xs:complexContent>
```

```
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:simpleType name="textSeparator">
    <xs:annotation>
        <xs:documentation>Max three characters to use as token or block separator</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:maxLength value="3"/>
    </xs:restriction>
</xs:simpleType>
<!-- ================================================== -->
<xs:simpleType name="decimalSeparator">
    <xs:annotation>
        <xs:documentation>One character to use as a decimal separator</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
        <xs:length value="1"/>
    </xs:restriction>
</xs:simpleType>
<!-- ================================================== -->
<!-- ================================================== -->
<xs:element name="BinaryBlock">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="swe:AbstractEncodingType">
                <xs:sequence>
                    <xs:element name="member" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:choice>
                                <xs:element name="Component">
                                    <xs:complexType>
                                        <xs:attribute name="ref" type="xs:token" use="required"/>
                                        <xs:attribute name="dataType" type="xs:anyURI" use="optional"/>
                                        <xs:attribute name="significantBits" type="xs:positiveInteger"
                                            use="optional"/>
                                        <xs:attribute name="bitLength" type="xs:positiveInteger"
                                            use="optional"/>
                                        <xs:attribute name="paddingBits-before" type="xs:nonNegativeInteger"
                                            use="optional" default="0"/>
                                        <xs:attribute name="paddingBits-after" type="xs:nonNegativeInteger"
                                            use="optional" default="0"/>
                                        <xs:attribute name="encryption" type="xs:anyURI" use="optional"/>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element name="Block">
                                    <xs:complexType>
                                        <xs:attribute name="ref" type="xs:token" use="required"/>
                                        <xs:attribute name="byteLength" type="xs:positiveInteger"
                                            use="optional"/>
                                        <xs:attribute name="paddingBytes-before"
                                            type="xs:nonNegativeInteger" use="optional" default="0"/>
                                        <xs:attribute name="paddingBytes-after" type="xs:nonNegativeInteger"
                                            use="optional" default="0"/>
                                        <xs:attribute name="encryption" type="xs:anyURI" use="optional"/>
                                        <xs:attribute name="compression" type="xs:anyURI" use="optional"/>
                                    </xs:complexType>
                                </xs:element>
                            </xs:choice>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="byteLength" type="xs:positiveInteger" use="optional"/>
                <xs:attribute name="byteEncoding" type="swe:byteEncoding" use="required"/>
                <xs:attribute name="byteOrder" type="swe:byteOrder" use="required"/>
            </xs:extension>
        </xs:complexContent>
```

```
    </xs:complexType>
</xs:element>
<!-- ================================================== -->
<xs:simpleType name="byteEncoding">
    <xs:restriction base="xs:token">
        <xs:enumeration value="base64"/>
        <xs:enumeration value="raw"/>
        <xs:enumeration value="hex"/>
    </xs:restriction>
</xs:simpleType>
<!-- ================================================== -->
<xs:simpleType name="byteOrder">
    <xs:restriction base="xs:token">
        <xs:enumeration value="bigEndian"/>
        <xs:enumeration value="littleEndian"/>
    </xs:restriction>
</xs:simpleType>
<!-- ================================================== -->
<xs:element name="MultiplexedStreamFormat" type="swe:MultiplexedStreamFormatType">
    <xs:annotation>
        <xs:documentation>
            Allows specification of the stream/packaging format used (ex: RTP, ASF, AAF, TML...)
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="MultiplexedStreamFormatType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractEncodingType">
            <xs:attribute name="type" type="xs:anyURI" use="required"/>
            <xs:attribute name="version" type="xs:string" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ........ -->
<xs:complexType name="MultiplexedStreamFormatPropertyType">
    <xs:sequence>
        <xs:element ref="swe:MultiplexedStreamFormat" minOccurs="0"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!-- ================================================== -->
<!-- ================================================== -->
<xs:element name="XMLBlock" type="swe:XMLBlockType">
    <xs:annotation>
        <xs:documentation>
            Carries the designator for an element implementing an XML-encoded data-type
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="XMLBlockType">
    <xs:annotation>
        <xs:documentation>
            Carries the designator for an element implementing an XML-encoded data-type
        </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="swe:AbstractEncodingType">
            <xs:attribute name="xmlElement" type="xs:QName">
                <xs:annotation>
                    <xs:documentation>
                        May be any XML Schema defined global element. Typically this will be swe:Array,
                        swe:Record, cv:CV_DiscreteCoverage, etc
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
```

```
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================= -->
</xs:schema>
```

B.7 curveTypes.xsd

Defines curves used throughout SWE.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <xs:annotation>
        <xs:documentation>
            Schema for defining different types of curves based on aggregate data types
        </xs:documentation>
    </xs:annotation>
    <!-- ================================================= -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:include schemaLocation="./aggregateTypes.xsd"/>
    <!-- ================================================= -->
    <!--                Curve Components                   -->
    <!-- ================================================= -->
    <xs:element name="Curve" type="swe:CurveType" substitutionGroup="swe:AbstractDataArray">
        <xs:annotation>
            <xs:documentation>
                Curve describing variations of a parameter vs. another quantity
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ........ -->
    <xs:complexType name="CurveType">
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataArrayType">
                <xs:sequence>
                    <xs:element name="elementType" type="swe:SimpleDataRecordPropertyType"/>
                    <xs:group ref="swe:EncodedValuesGroup" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ........ -->
    <xs:complexType name="CurvePropertyType">
        <xs:annotation>
            <xs:documentation>
                Curve is a data-type so usually appears "by value" rather than by reference.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="swe:Curve"/>
        </xs:sequence>
    </xs:complexType>
    <!-- ================================================= -->
    <xs:element name="NormalizedCurve" type="swe:NormalizedCurveType"
        substitutionGroup="swe:AbstractDataRecord"/>
    <xs:complexType name="NormalizedCurveType">
        <xs:complexContent>
            <xs:extension base="swe:AbstractDataRecordType">
                <xs:sequence>
```

```
                    <xs:element name="inputGain" type="swe:QuantityPropertyType" minOccurs="0"/>
                    <xs:element name="inputBias" type="swe:QuantityPropertyType" minOccurs="0"/>
                    <xs:element name="outputGain" type="swe:QuantityPropertyType" minOccurs="0"/>
                    <xs:element name="outputBias" type="swe:QuantityPropertyType" minOccurs="0"/>
                    <xs:element name="interpolationMethod" type="swe:CategoryPropertyType" minOccurs="0"/>
                    <xs:element name="extrapolationMethod" type="swe:CategoryPropertyType" minOccurs="0"/>
                    <xs:element name="function" type="swe:CurvePropertyType"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================ -->
    <!--              Data Component Groups               -->
    <!-- ================================================ -->
    <xs:group name="Curves">
        <xs:choice>
            <xs:element ref="swe:Curve"/>
            <xs:element ref="swe:NormalizedCurve"/>
        </xs:choice>
    </xs:group>
</xs:schema>
```

B.8 positionTypes.xsd.

The positionTypes.xsd schema defines positional data types throughout the Sensor Web Enablement encodings and services.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:gml="http://www.opengis.net/gml"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <xs:annotation>
        <xs:documentation>
            Schema for defining position data (location, orientation, etc...) based on aggregate data types
        </xs:documentation>
    </xs:annotation>
    <!-- ================================================ -->
    <xs:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <xs:include schemaLocation="./aggregateTypes.xsd"/>
    <!-- ================================================ -->
    <xs:element name="Position" type="swe:PositionType" substitutionGroup="swe:AbstractDataRecord">
        <xs:annotation>
            <xs:documentation>
                Position is given as a group of Vectors/Matrices, each of which can specify location, orientation,
                velocity, angularVelocity, acceleration or angularAcceleration or a combination of those in a
                composite state Vector. Each Vector can have a separate frame of reference or inherit the frame
                from the parent Position object.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ..... -->
    <xs:complexType name="PositionType">
        <xs:complexContent>
            <xs:extension base="swe:AbstractVectorType">
                <xs:sequence>
                    <xs:element name="time" type="swe:TimePropertyType" minOccurs="0"/>
                    <xs:element name="location" type="swe:VectorPropertyType" minOccurs="0"/>
                    <xs:element name="orientation" type="swe:VectorOrSquareMatrixPropertyType"
                        minOccurs="0"/>
                    <xs:element name="velocity" type="swe:VectorPropertyType" minOccurs="0"/>
                    <xs:element name="angularVelocity" type="swe:VectorOrSquareMatrixPropertyType"
                        minOccurs="0"/>
```

```
                <xs:element name="acceleration" type="swe:VectorPropertyType" minOccurs="0"/>
                <xs:element name="angularAcceleration" type="swe:VectorOrSquareMatrixPropertyType"
                    minOccurs="0"/>
                <xs:element name="state" type="swe:VectorOrSquareMatrixPropertyType" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:complexType name="AbstractMatrixType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataArrayType">
            <xs:attribute name="referenceFrame" type="xs:anyURI" use="optional">
                <xs:annotation>
                    <xs:documentation>
                        Points to a spatial reference frame definition. Coordinates of the vector will be
                        expressed in this reference frame
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="localFrame" type="xs:anyURI" use="optional">
                <xs:annotation>
                    <xs:documentation>
                        Specifies the spatial frame which location and/or orientation is given by the enclosing
                        vector
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================ -->
<xs:element name="SquareMatrix" type="swe:SquareMatrixType"
    substitutionGroup="swe:AbstractDataArray">
    <xs:annotation>
        <xs:documentation>
            This is a square matrix (so the size is the square of one dimension) which is a DataArray of
            Quantities.  It has a referenceFrame in which the matrix components are described
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="SquareMatrixType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractMatrixType">
            <xs:sequence>
                <xs:element name="elementType" type="swe:QuantityPropertyType"/>
                <xs:group ref="swe:EncodedValuesGroup" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ........ -->
<xs:complexType name="VectorOrSquareMatrixPropertyType">
    <xs:annotation>
        <xs:documentation>
            Vector/SquareMatrix is a data-type so usually appears "by value" rather than by reference.
        </xs:documentation>
    </xs:annotation>
    <xs:choice>
        <xs:element ref="swe:Vector"/>
        <xs:element ref="swe:SquareMatrix"/>
    </xs:choice>
</xs:complexType>
<!-- ================================================ -->
<!-- Areas and Envelopes -->
<!-- ================================================ -->
```

```
<xs:element name="GeoLocationArea" substitutionGroup="swe:AbstractDataRecord">
    <xs:annotation>
        <xs:documentation>Area used to define bounding boxes</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="swe:AbstractVectorType">
                <xs:sequence>
                    <xs:element name="member" type="swe:EnvelopePropertyType"
                        maxOccurs="unbounded">
                        <xs:annotation>
                            <xs:documentation>Is this an aggregate geometry?</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="name" type="xs:token"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- ============================================== -->
<xs:element name="Envelope" type="swe:EnvelopeType" substitutionGroup="swe:AbstractDataRecord">
    <xs:annotation>
        <xs:documentation>
            Envelope described using two vectors specifying lower and upper corner points.
            This is typically use to define rectangular bounding boxes in any coordinate system.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- ........ -->
<xs:complexType name="EnvelopeType">
    <xs:complexContent>
        <xs:extension base="swe:AbstractVectorType">
            <xs:sequence>
                <xs:element name="time" type="swe:TimeRangePropertyType" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>
                            Optionally provides time range during which this bounding envelope applies
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="lowerCorner" type="swe:VectorPropertyType"/>
                <xs:element name="upperCorner" type="swe:VectorPropertyType"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ........ -->
<xs:complexType name="EnvelopePropertyType">
    <xs:sequence minOccurs="0">
        <xs:element ref="swe:Envelope"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xs:complexType>
<!--       Composite/Aggregate Components - Vectors and Matrices          -->
<!-- ============================================== -->
<xs:complexType name="AbstractVectorType" abstract="true">
    <xs:complexContent>
        <xs:extension base="swe:AbstractDataRecordType">
            <xs:attribute name="referenceFrame" type="xs:anyURI" use="optional">
                <xs:annotation>
                    <xs:documentation>
                        Points to a spatial reference frame definition. Coordinates of the vector will be
                        expressed in this reference frame
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
```

```
                <xs:attribute name="localFrame" type="xs:anyURI" use="optional">
                    <xs:annotation>
                        <xs:documentation>
                            Specifies the spatial frame which location and/or orientation is given by the enclosing
                            vector
                        </xs:documentation>
                    </xs:annotation>
                </xs:attribute>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ================================================ -->
    <xs:element name="Vector" type="swe:VectorType" substitutionGroup="swe:AbstractDataRecord">
        <xs:annotation>
            <xs:documentation>
                A Vector is a special type of DataRecord that takes a list of numerical scalars called coordinates.
                The Vector has a referenceFrame in which the coordinates are expressed
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!-- ................ -->
    <xs:complexType name="VectorType">
        <xs:complexContent>
            <xs:extension base="swe:AbstractVectorType">
                <xs:sequence>
                    <xs:element name="coordinate" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:group ref="swe:AnyNumerical" minOccurs="0"/>
                            <xs:attribute name="name" type="xs:token" use="required"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- ........ -->
    <xs:complexType name="VectorPropertyType">
        <xs:annotation>
            <xs:documentation>
                Vector is a data-type so usually appears "by value" rather than by reference. However, by
                reference is still useful when objects, for instance, share a location.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence minOccurs="0">
            <xs:element ref="swe:Vector"/>
        </xs:sequence>
        <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
    </xs:complexType>
    <!-- ================================================ -->
    <!-- ================================================ -->
</xs:schema>
```

B.9 temporalAggregates.xsd.

Defines time geometric complex, time aggregates, and time grids.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <annotation>
        <documentation>
```

```
            temporalAggregates.xsd
            time geometric complex, time aggregates and time grids

            Copyright (c) 2007 Open Geospatial Consortium - see http://www.opengeospatial.org/about/?page=ipr

        </documentation>
    </annotation>
    <!-- ============================================================================ -->
    <!-- bring in other schemas -->
    <import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <!-- ============================================================================ -->
    <!-- ============================================================ -->
    <!-- ============================================================ -->
    <simpleType name="TimeValueList">
        <annotation>
            <documentation>Compact list of time instants, following gml:posList pattern. </documentation>
        </annotation>
        <list itemType="gml:TimePositionUnion"/>
    </simpleType>
    <!-- ============================================================================ -->
    <complexType name="TimePositionListType">
        <annotation>
            <documentation>
                TimePositionList instances hold a sequence of time positions within the same frame.
            </documentation>
        </annotation>
        <simpleContent>
            <extension base="swe:TimeValueList">
                <attribute name="frame" type="anyURI" use="optional" default="#ISO-8601"/>
                <attribute name="calendarEraName" type="string" use="optional"/>
                <attribute name="indeterminatePosition" type="gml:TimeIndeterminateValueType"
                    use="optional"/>
                <attribute name="count" type="positiveInteger" use="optional">
                    <annotation>
                        <documentation>
                            "count" allows to specify the number of direct positions in the list.
                        </documentation>
                    </annotation>
                </attribute>
            </extension>
        </simpleContent>
    </complexType>
    <!-- ============================================================ -->
    <!-- ===== Missing property types ====== -->
    <!-- ============================================================================ -->
    <complexType name="TimeGeometricPrimitivePropertyType">
        <annotation>
            <documentation>Property type not provided by GML</documentation>
        </annotation>
        <sequence minOccurs="0">
            <element ref="gml:_TimeGeometricPrimitive"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </complexType>
    <!-- ============================================================ -->
    <!-- ===== Time geometric complexes ====== -->
    <!-- ============================================================================ -->
    <complexType name="TimeGeometricComplexType">
        <annotation>
            <documentation>
                Explicit implementation of ISO 19108 TM_GeometricComplex - a self-consistent set of
                TimeInstants and TimePeriods
            </documentation>
        </annotation>
        <complexContent>
            <extension base="gml:AbstractTimeComplexType">
```

```
            <sequence>
                <element name="primitive" type="swe:TimeGeometricPrimitivePropertyType"
                    maxOccurs="unbounded">
                    <annotation>
                        <documentation>Reference to an identified time primitive</documentation>
                    </annotation>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<!-- ..... -->
<element name="TimeGeometricComplex" type="swe:TimeGeometricComplexType"
    substitutionGroup="gml:_TimeComplex">
    <annotation>
        <documentation>
            Explicit implementation of ISO 19108 TM_GeometricComplex - a self-consistent set of
            TimeInstants and TimePeriods
        </documentation>
    </annotation>
</element>
<!-- ..... -->
<complexType name="TimeGeometricComplexPropertyType">
    <sequence minOccurs="0">
        <element ref="swe:TimeGeometricComplex"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
<!-- ================================== -->
<!-- ===== Time aggregates ====== -->
<!-- ================================================================== -->
<!-- ===== Explicit time aggregates ====== -->
<!-- ================================================================== -->
<complexType name="TimeObjectPropertyType">
    <annotation>
        <documentation>Property type not provided by GML</documentation>
    </annotation>
    <sequence minOccurs="0">
        <element ref="gml:_TimeObject"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
<!-- ================================================================== -->
<complexType name="TimeAggregateType">
    <annotation>
        <documentation>
            an arbitrary set of TimeObjects, often TimeInstants and TimePeriods
        </documentation>
    </annotation>
    <complexContent>
        <extension base="gml:AbstractTimeObjectType">
            <sequence>
                <element name="member" type="swe:TimeObjectPropertyType" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<!-- ..... -->
<element name="TimeAggregate" type="swe:TimeAggregateType" substitutionGroup="gml:_TimeObject">
    <annotation>
        <documentation>
            an arbitrary set of TimeObjects, often TimeInstants and TimePeriods
        </documentation>
    </annotation>
</element>
<!-- ..... -->
<complexType name="TimeAggregatePropertyType">
```

```
        <sequence minOccurs="0">
            <element ref="swe:TimeAggregate"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </complexType>
    <!-- ================================== -->
    <!-- ===== Implicit time aggregates ====== -->
    <!-- ================================================================== -->
    <complexType name="TimeGridType">
        <annotation>
            <documentation>
                A set of uniformly spaced time instants described using an implicit notation.
                Follow pattern of (ISO 19123) spatial grids:
                    these have (dimension,axisName,extent(,origin,offsetVector))
                For temporal case, dimension is fixed (1), axisName is fixed ("time")
            </documentation>
        </annotation>
        <complexContent>
            <extension base="gml:AbstractTimeComplexType">
                <sequence>
                    <element name="extent" type="swe:TimeGridEnvelopePropertyType"/>
                    <choice>
                        <element name="originPos" type="gml:TimePositionType">
                            <annotation>
                                <documentation>Simple-content time position</documentation>
                            </annotation>
                        </element>
                        <element name="origin" type="gml:TimeInstantPropertyType">
                            <annotation>
                                <documentation>Reference to an identified time instant</documentation>
                            </annotation>
                        </element>
                    </choice>
                    <choice>
                        <element name="offsetDuration" type="duration">
                            <annotation>
                                <documentation>
                                    XML Schema built-in simple type for duration: e.g.
                                    P1Y (1 year)
                                    P1M (1 month)
                                    P1DT12H (1 day 12 hours)
                                    PT5M (5 minutes)
                                    PT0.007S (7 milliseconds)
                                </documentation>
                            </annotation>
                        </element>
                        <element name="offsetInterval" type="gml:TimeIntervalLengthType">
                            <annotation>
                                <documentation>
                                    representation of the ISO 11404 model of a time interval length: e.g.
                                    value=1, unit="year"
                                    value=1, unit="other:month" (or see next)
                                    value=1, unit="year" radix="12" factor="1" (1/12 year)
                                    value=1.5, unit="day"
                                    value=36, unit="hour"
                                    value=5, unit="minute"
                                    value=7, unit="second" radix="10" factor="3" (7 milliseconds)
                                </documentation>
                            </annotation>
                        </element>
                    </choice>
                    <element name="duration" type="duration" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <!-- ..................... -->
```

```
<element name="TimeGrid" type="swe:TimeGridType" abstract="true"
    substitutionGroup="gml:_TimeComplex">
    <annotation>
        <documentation>
            A set of uniformly spaced time instants described using an implicit notation
            Follow pattern of (ISO 19123) spatial grids:
                these have (dimension,axisName,extent(,origin,offsetVector))
            For temporal case, dimension is fixed (1), axisName is fixed ("time")
        </documentation>
    </annotation>
</element>
<!-- ..... -->
<complexType name="TimeGridPropertyType">
    <sequence minOccurs="0">
        <element ref="swe:TimeInstantGrid"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
<!-- ========================================================== -->
<complexType name="TimeGridEnvelopePropertyType">
    <sequence>
        <element name="TimeGridEnvelope">
            <annotation>
                <documentation>Grid extent specified in grid coordinates - i.e. 2 integers</documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element name="low" type="integer"/>
                    <element name="high" type="integer"/>
                </sequence>
            </complexType>
        </element>
    </sequence>
</complexType>
<!-- ================================================================ -->
<complexType name="TimeInstantGridType">
    <annotation>
        <documentation>Extend time instant grid with window size property</documentation>
    </annotation>
    <complexContent>
        <extension base="swe:TimeGridType"/>
    </complexContent>
</complexType>
<!-- ..... -->
<element name="TimeInstantGrid" type="swe:TimeInstantGridType" substitutionGroup="swe:TimeGrid">
    <annotation>
        <documentation>
            A set of uniformly spaced time instants described using an implicit notation
        </documentation>
    </annotation>
</element>
<!-- ..... -->
<complexType name="TimeInstantGridPropertyType">
    <sequence minOccurs="0">
        <element ref="swe:TimeInstantGrid"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
<!-- ================================================================ -->
<complexType name="TimeIntervalGridType">
    <annotation>
        <documentation>Extend time instant grid with window size property</documentation>
    </annotation>
    <complexContent>
        <extension base="swe:TimeGridType">
            <sequence>
                <choice>
```

```
                        <element name="windowDuration" type="duration">
                            <annotation>
                                <documentation>XML Schema built-in simple type for duration</documentation>
                            </annotation>
                        </element>
                        <element name="windowInterval" type="gml:TimeIntervalLengthType">
                            <annotation>
                                <documentation>
                                    representation of the ISO 11404 model of a time interval length
                                </documentation>
                            </annotation>
                        </element>
                    </choice>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <!-- ..... -->
    <element name="TimeIntervalGrid" type="swe:TimeIntervalGridType" substitutionGroup="swe:TimeGrid">
        <annotation>
            <documentation>
                A set of uniformly spaced time intervals described using an implicit notation
            </documentation>
        </annotation>
    </element>
    <!-- ..... -->
    <complexType name="TimeIntervalGridPropertyType">
        <sequence minOccurs="0">
            <element ref="swe:TimeIntervalGrid"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </complexType>
    <!-- ================================================================== -->
</schema>
```

B.10 phenomenon.xsd.

Defines phenomenon entries.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
    <annotation>
        <documentation>
            phenomenon.xsd

            A GML conformant schema for definitions of phenomena, per Annex C of OM specification

            Copyright (c) 2007 Open Geospatial Consortium - see http://www.opengeospatial.org/about/?page=ipr
        </documentation>
    </annotation>
    <!-- ================================================================== -->
    <!-- bring in other schemas -->
    <import namespace="http://www.opengis.net/gml"
        schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    <include schemaLocation="./basicTypes.xsd"/>
    <!-- ================================================================== -->
    <complexType name="PhenomenonType">
        <annotation>
            <documentation>
                Basic Phenomenon definition, and head of substitution group of specialized phenomenon defs.
                gml:description may be used for a more extensive description of the semantics, with a link to a
```

```
                definitive version (if available).
                gml:name should be used for the name or label.

                Note: In GML 3.2 the gml:identifier element should be used for the identifier assigned by or
                preferred by the data provider.
                This identifier will typically be in the form of a URN, for example following the OGC URN scheme
                e.g. urn:ogc:def:property:OGC:Age
            </documentation>
        </annotation>
        <complexContent>
            <extension base="gml:DefinitionType"/>
        </complexContent>
    </complexType>
    <!-- ..... -->
    <element name="Phenomenon" type="swe:PhenomenonType" substitutionGroup="gml:Definition">
        <annotation>
            <documentation>
                Basic Phenomenon definition, and head of substitution group of specialized phenomenon defs.
            </documentation>
        </annotation>
    </element>
    <!-- ..... -->
    <complexType name="PhenomenonPropertyType">
        <sequence minOccurs="0">
            <element ref="swe:Phenomenon"/>
        </sequence>
        <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </complexType>
    <!-- ========================================================================== -->
    <complexType name="ConstrainedPhenomenonType">
        <annotation>
            <documentation>
                A scalar Phenomenon is defined by adding constraints to an existing property.
            </documentation>
        </annotation>
        <complexContent>
            <extension base="swe:PhenomenonType">
                <sequence>
                    <element name="base" type="swe:PhenomenonPropertyType">
                        <annotation>
                            <documentation>
                                Property that forms the basis for generating a set of more refined Phenomena; e.g.
                                Chemical Composition, Radiance
                            </documentation>
                        </annotation>
                    </element>
                    <element name="otherConstraint" type="string" minOccurs="0" maxOccurs="unbounded">
                        <annotation>
                            <documentation>
                                Constraints that cannot be expressed as values of an orthogonal/helper
                                phenomenon
                            </documentation>
                        </annotation>
                    </element>
                    <element name="singleConstraint" type="swe:TypedValuePropertyType" minOccurs="0"
                        maxOccurs="unbounded">
                        <annotation>
                            <documentation>
                                Constraint expressed as a value or range of an orthogonal/helper phenomenon
                            </documentation>
                        </annotation>
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <!-- ..... -->
```

```
<element name="ConstrainedPhenomenon" type="swe:ConstrainedPhenomenonType"
    substitutionGroup="swe:Phenomenon">
    <annotation>
        <documentation>
            Description of a scalar Phenomenon defined by adding constraints to a property previously defined
            elsewhere.
        </documentation>
    </annotation>
</element>
<!-- ========================================================================== -->
<!-- ========================================================================== -->
<complexType name="CompoundPhenomenonType" abstract="true">
    <annotation>
        <documentation>
            Description of a set of Phenomena.
            CompoundPhenomenon is the abstract head of a substitution group of specialized compound
            phenomena
        </documentation>
    </annotation>
    <complexContent>
        <extension base="swe:PhenomenonType">
            <attribute name="dimension" type="positiveInteger" use="required">
                <annotation>
                    <documentation>The number of components in the tuple</documentation>
                </annotation>
            </attribute>
        </extension>
    </complexContent>
</complexType>
<!-- ..................................... -->
<element name="CompoundPhenomenon" type="swe:CompoundPhenomenonType" abstract="true"
    substitutionGroup="swe:Phenomenon">
    <annotation>
        <documentation>
            Description of a set of Phenomena.
            CompoundPhenomenon is the abstract head of a substitution group of specialized compound
            phenomena
        </documentation>
    </annotation>
</element>
<!-- ========================================================================== -->
<complexType name="CompositePhenomenonType">
    <annotation>
        <documentation>
            A Phenomenon defined as a set of explicitly enumerated components which may or may not be
            related to one another
        </documentation>
    </annotation>
    <complexContent>
        <extension base="swe:CompoundPhenomenonType">
            <sequence>
                <element name="base" type="swe:PhenomenonPropertyType" minOccurs="0">
                    <annotation>
                        <documentation>
                            Optional phenomenon that forms the basis for generating more specialized
                            composite Phenomenon by adding more components
                        </documentation>
                    </annotation>
                </element>
                <element name="component" type="swe:PhenomenonPropertyType"
                    maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<!-- ..................................... -->
<element name="CompositePhenomenon" type="swe:CompositePhenomenonType"
```

```
        substitutionGroup="swe:CompoundPhenomenon">
        <annotation>
            <documentation>
                A Phenomenon defined as a set of explicitly enumerated components which may or may not be
                related to one another
            </documentation>
        </annotation>
    </element>
    <!-- ========================================================================== -->
    <complexType name="PhenomenonSeriesType">
        <annotation>
            <documentation>
                A phenomenon defined as a base property convolved with a set of constraints
                The set of constraints may be either
                    * an explicit set of soft-typed measures, intervals and categories
                    * one or more lists of soft-typed measures, intervals and categories
                    * one or more sequences of soft-typed measures and intervals
            </documentation>
        </annotation>
        <complexContent>
            <extension base="swe:CompoundPhenomenonType">
                <sequence>
                    <element name="base" type="swe:PhenomenonPropertyType">
                        <annotation>
                            <documentation>
                                Phenomenon that forms the basis for generating a set of more refined
                                Phenomena; e.g. Chemical Composition, Radiance
                            </documentation>
                        </annotation>
                    </element>
                    <element name="constraintList" type="swe:TypedValueListPropertyType"
                        maxOccurs="unbounded">
                        <annotation>
                            <documentation>
                                A set of values of some secondary property that constraints the basePhenomenon
                                to generate a Phenomenon set.
                                If more than one set of constraints are possible, then these are applied
                                simultaneously to generate
                            </documentation>
                        </annotation>
                    </element>
                    <element name="otherConstraint" type="string" minOccurs="0" maxOccurs="unbounded">
                        <annotation>
                            <documentation>Other constraints are described in text</documentation>
                        </annotation>
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <!-- ..................................... -->
    <element name="PhenomenonSeries" type="swe:PhenomenonSeriesType"
        substitutionGroup="swe:CompoundPhenomenon">
        <annotation>
            <documentation>
                A phenomenon defined as a base property convolved with a set of constraints
                The set of constraints may be either
                    * an explicit set of soft-typed measures, intervals and categories
                    * one or more lists of soft-typed measures, intervals and categories
                    * one or more sequences of soft-typed measures and intervals
            </documentation>
        </annotation>
    </element>
    <!-- ========================================================================== -->
    <!-- ========================================================================== -->
</schema>
```

B.11 xmlData.xsd.

Defines a data encoding that supports XML tag-separated data values.

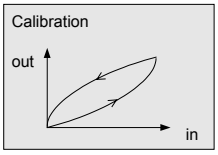
```
<?xml version="1.0"?>
<schema targetNamespace="http://www.opengis.net/swe/1.0"
    xmlns:swe="http://www.opengis.net/swe/1.0"
    xmlns="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="pre-release">
    <annotation>
        <documentation>
            A basic schema for data aggregates (Records and Arrays), using terminology consistent with ISO
            11404.

            A schema for the description of the record is given in recordType.xsd</documentation>
    </annotation>
    <!-- ============== -->
    <!-- ============== -->
    <element name="Item">
        <annotation>
            <documentation>An Item is an item of data of any type</documentation>
        </annotation>
    </element>
    <!-- ......  ...... -->
    <complexType name="ItemPropertyType">
        <sequence>
            <element ref="swe:Item"/>
        </sequence>
    </complexType>
    <!-- ============== -->
    <!-- ============== -->
    <complexType name="RecordType">
        <annotation>
            <documentation>A record is a list of fields</documentation>
        </annotation>
        <sequence>
            <element name="field" type="swe:ItemPropertyType" maxOccurs="unbounded">
                <annotation>
                    <documentation>A Record/field contains an item of data</documentation>
                </annotation>
            </element>
        </sequence>
        <attribute name="RS" type="anyURI" use="optional">
            <annotation>
                <documentation>Optional pointer to record-type schema</documentation>
            </annotation>
        </attribute>
        <attribute name="fieldCount" type="positiveInteger" use="optional">
            <annotation>
                <documentation>Optional count of the number of fields in the record. </documentation>
            </annotation>
        </attribute>
    </complexType>
    <!-- ......  ...... -->
    <element name="Record" type="swe:RecordType">
        <annotation>
            <documentation>A record is a list of fields</documentation>
        </annotation>
    </element>
    <!-- ......  ...... -->
    <complexType name="RecordPropertyType">
        <sequence>
            <element ref="swe:Record"/>
        </sequence>
    </complexType>
    <!-- ============== -->
```

```
    <!-- ============== -->
    <complexType name="ArrayType">
        <annotation>
            <documentation>An array is an indexed set of records of homogeneous type</documentation>
        </annotation>
        <sequence>
            <element name="element" type="swe:XMLDataPropertyType" maxOccurs="unbounded">
                <annotation>
                    <documentation>An Array/element contains an Item or a Record or an Array</documentation>
                </annotation>
            </element>
        </sequence>
        <attribute name="RS" type="anyURI" use="optional">
            <annotation>
                <documentation>
                    Optional pointer to the record-type schema. This should be used when the elements of the
                    array are Records
                </documentation>
            </annotation>
        </attribute>
        <attribute name="elementCount" type="positiveInteger" use="optional">
            <annotation>
                <documentation>Optional count of the number of elements in the array. </documentation>
            </annotation>
        </attribute>
    </complexType>
    <!-- ......  ...... -->
    <element name="Array" type="swe:ArrayType">
        <annotation>
            <documentation>An array is an indexed set of records of homogeneous type</documentation>
        </annotation>
    </element>
    <!-- ......  ...... -->
    <complexType name="ArrayPropertyType">
        <sequence>
            <element ref="swe:Array"/>
        </sequence>
    </complexType>
    <!-- ============== -->
    <complexType name="XMLDataPropertyType">
        <annotation>
            <documentation>Choice of Item or Record or Array - used in composing Arrays</documentation>
        </annotation>
        <group ref="swe:XMLData"/>
    </complexType>
    <!-- ============== -->
    <group name="XMLData">
        <annotation>
            <documentation>
                Convenience group that bundles all the soft-typed XML-encoded aggregates into a choice group
            </documentation>
        </annotation>
        <choice>
            <element ref="swe:Item"/>
            <element ref="swe:Record"/>
            <element ref="swe:Array"/>
        </choice>
    </group>
    <!-- ============== -->
</schema>
```

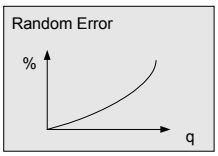
Annex C. Annex C: Model for Detector (Informative)

This annex presents a model to be profiled in SensorML and to be used for describing a wide range of detectors whether they are in-situ or remote. This model, inspired by TML concepts as well expert advice from various communities (see references), covers a very wide range of sensors by providing a list of generic response parameters applicable to any detector type. A *Detector* is defined here as a specific process responsible for sampling and converting a scalar input signal to a scalar output signal. These parameters include the following:



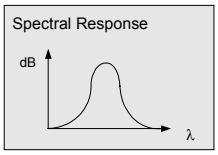
Calibration Curve

Gives the mapping of input to output values for a steady state regime. Two curves are used to describe a Hysteretic behavior.



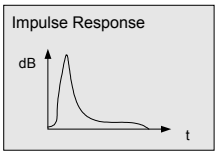
Random Error Curve

Gives the relative measurement error versus the input value itself or any other environmental quantity such as temperature.



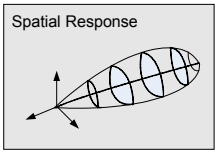
Spectral Response Curve

Specifies dynamic characteristics of the detector in the frequency domain. It gives the sensitivity of the detector versus the frequency or wavelength of the input signal.



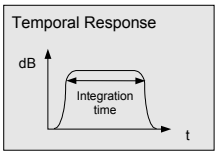
Impulse Response Curve

Specifies dynamic characteristics of the detector in the time domain. It represents the normalized output of the detector for an impulse (Δ function) input.



Spatial Response Curve(s)

Gives the sensitivity of the detector relative to spatial coordinates (location of the source, or orientation of the incoming signal, e.g., point spread function, polarization)



Temporal Response Curve

Gives the sensitivity of the detector relative to a temporal coordinate frame (e.g., sampling time). This is a more descriptive form of the integration time.

All these parameters are conditional, which means that each curve is valid under a certain number of conditions. A common condition is the calibration time at which these parameters are derived.

Several other scalar parameters can also be specified, most of which are used to describe the same aspect of the response than some of the curves stated above (curves can be derived from these parameters). These parameters are also conditional and include:

Accuracy Value (corresponds to the error curve)

Specifies the accuracy of measurement, by using either an absolute (in the unit of measure of the output) or relative (in percent) value.

Latency Time

Specifies the overall delay between the input sampling time and the availability of the output value (this includes processing time if any).

Integration Time (corresponds to the temporal response curve)

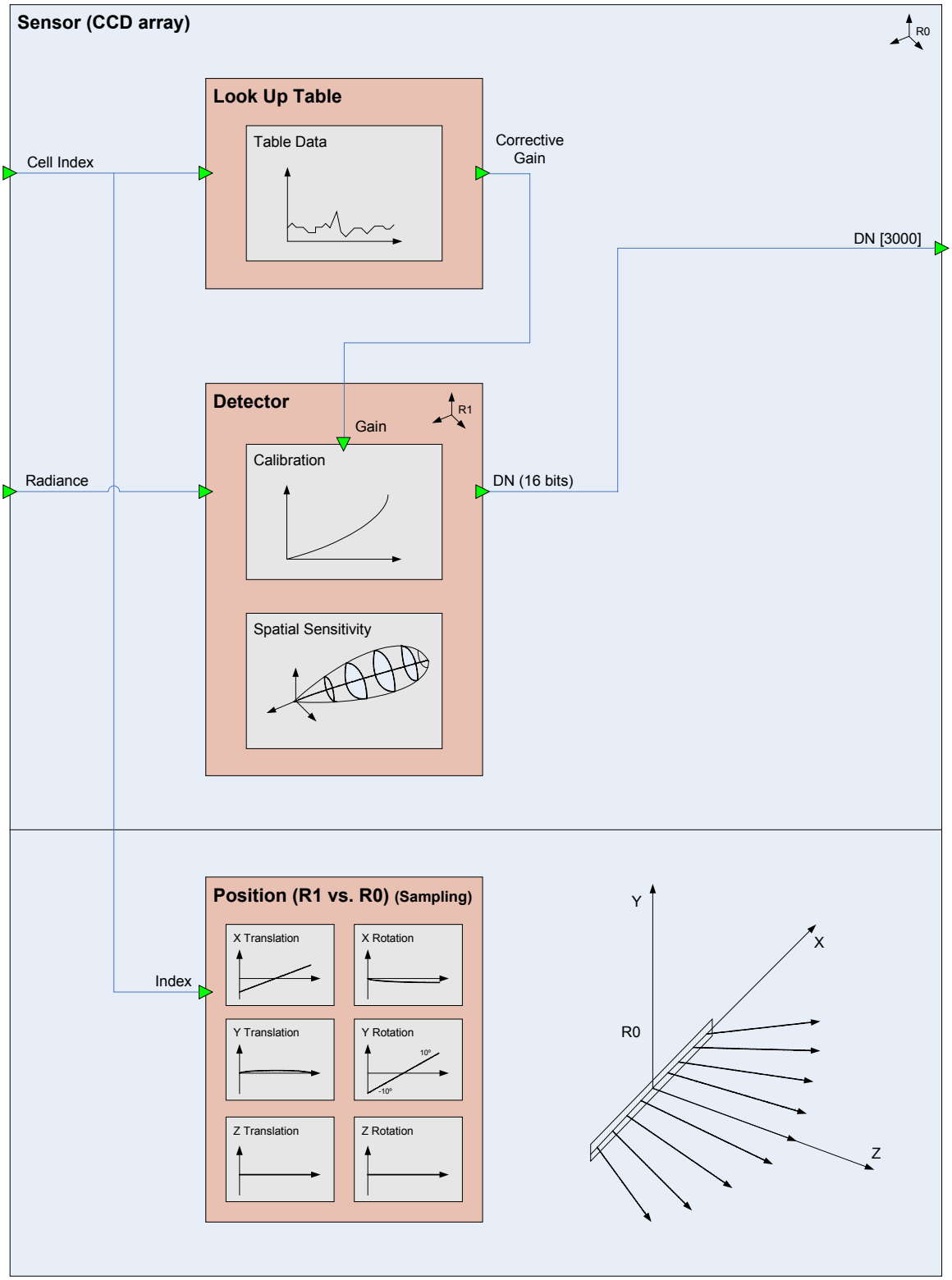
Describes the period of influence of the detector during which the input phenomenon signal is integrated (e.g., shutter speed).

Integration Space (corresponds to the spatial response curve)

Describes the volume of influence of the detector from which the input phenomenon signal is integrated. (e.g., this is given by a list of ifov angles along different axes)

Note that in order to be meaningful, *Integration Time* and *Integration Space* need to be specified at a particular 'power level' typically expressed in dB. This is achieved by specifying a condition along with the value.

This *Detector* model, encoded as a specific *Component* in SensorML, can also be used within a *System* to generate composite measurements (e.g., a weather station would be a *System* containing a thermometer (temperature detector), a barometer (pressure detector) and an anemometer (wind speed detector)). A *System* or *Sensor* can also be used as a wrapper around one or several *Detectors* to create more complex *Detector Arrays* (e.g., in a CCD camera, each CCD cell would be a *Detector* and the whole *Sensor* is an array of perhaps 1600x1200 detectors). SensorML provides index based mechanism to tweak individual detector parameters and position without having to describe each detector (1920000 in this case!) separately. This is shown on the following diagram:



Annex D: History of SensorML (Informative)

The SensorML was originally conceived as a sensor description language that would aid in the processing and geolocation of multiple sensors. Being driven by the remote sensing community, the original focus was on providing properties to assist in the geolocation of individual pixels within scanners and cameras.

The concept for the standardized description files for all aspects of sensor geolocation was first proposed by the planetary science community and was implemented in the SPICE software system, that was developed and maintained by the Navigation and Ancillary Information Facility (NAIF) at NASA JPL. Written in FORTRAN and utilized by the planetary science community for nearly every mission since Galileo, SPICE has proved beneficial by reducing redundant programming for each mission, and by providing additional benefits not previous experienced before the implementation of SPICE.

In 1993, the University of Alabama in Huntsville (UAH) in cooperation with NASA JPL, implemented and tested the SPICE concepts for application within the Earth observation community. Termed the NASA EOS Interuse Experiment, this research effort focused on improving the integration of multiple sensor data by avoiding the need to distribute data as incompatible map projection grids. The results of the Interuse Experiment proved that a common sensor description with common processing software for Earth observation sensors was well within our abilities, and provided perhaps even more significant benefits to the Earth observation than had been experienced within the planetary community. With subsequent funding, the UAH VisAnalysis System Technologies (VAST) team has been implementing a more lightweight geolocation and processing environment directed principally for the Earth observation community and developed in Java.

In April 1998, the Global Mapping Task Team (GMTT) within Committee for Earth Observation Satellites (CEOS) released the following recommendation:

> **April 1998 - Recommendation to CEOS: Interoperability of Multiple Space Agency Sensor Data from the Global Mapping Task Team – Bernried, Germany**
>
> **Definition of Problem:** There is an increasing realization by Earth observation scientists that data from space-borne sensors are not adequately nor easily georeferenced to meet their requirements. The consequence of this is that it is currently extremely difficult or impossible to combine data from different space-borne sensors or ground-based data. The first impediment is the lack of adequate, publicly available data on the spatial-temporal extents of data from space-borne sensors.
>
> **Recommendation:** We, the CEOS Global Mapping Task Team, recommend that the space agencies seriously consider the production, storage, public access, and interoperability of adequate data for describing the dynamics and geometry of the sensor system. These data might include satellite position (ephemeris), satellite rotations (attitude), sensor model (dynamics, geometry, and calibration), relevant planet models, and spacecraft clock model. This data should be made available in real-time. There should also be an effort to provide publicly available software to ingest the above data using a common API. Any

recommendations for the most appropriate propagation model should be adequate documented or algorithms provided.

In September, 1998, Mike Botts introduced the beginnings of a "Sensor Description Format" to the CEOS GMTT and received recommendations to consider XML as a description framework. In September 1999, the initial XML-based version of the SensorML was introduced to the CEOS GMTT by Dr. Botts. In March 2000, Dr. Botts was awarded a contract through the NASA AIST program to complete, implement, and test the SensorML (funding was actually received in December 2000). Initial focus of the SensorML was on the geolocation and description of remote sensing instruments, with minor attention given to measurement characteristics such as radiometry [Botts, 2001].

In Fall 2000, Dr. Liping Di (a CEOS GMTT member) proposed to the ISO TC211 Committee to establish a standard sensor and data model framework. This was approved in December 2000 and Dr. Botts was asked to serve as a team member. The first meeting was scheduled for June 2001. It is expected that the SensorML will both influence and be influenced by the ISO activities. The intent is that the SensorML will be a compliant implementation of the ISO standard.

In March 2001, Dr. Botts was accepted to participate in the OpenGIS Consortium (OGC) Military Pilot Project (MPP-1) with an emphasis on introducing the concepts of the SensorML into the OGC Sensor Web Enablement (SWE) initiative.

In September 2001, the OGC Interoperability Program (IP) initiated the Open Web Services (OWS) project with one of three threads focused on the initial design and testing of SWE concepts. Within that initiative, the SensorML provides a key component for describing the characteristics and capabilities of any sensor, whether the sensor is designed for in-situ or remote observation. Because of the focus on in-situ sensor within OWS 1.1, the initial phase of this activity was to drive the development of the SensorML into several new directions, including:

- Generalizing the structure and grammar to support both in-situ and remote sensors
- Providing more generalized and more robust support for describing measurement characteristics, regardless of whether the sensor measures radiation, chemical concentration, velocity, temperature, or any other physical phenomena
- Further migration toward an XML schema, with inheritance from other schema, such as OGC GML
- Providing more robust support for non-scanning sensors, such as profilers and frame cameras

In April 2002, the SensorML document OGC 02-026 was approved as a Public Discussion Paper [OGC 02-026].

From May to December 2002, OGC IP conducted a follow-up OWS (OWS 1.2) project. With regard to SWE activities, the focus was extended to remote sensors on static or dynamic platforms. With regard to SensorML, the activities related to this project included:

- General refinement of SensorML through incorporation of schema components from ISO 19115
- Determined the role of GML within SensorML
- Extension and refinement of support for multiple sensor geolocation models, including scanner/profilers, frame cameras, and rapid positioning coordinates (RPC).
- Support for dynamic platforms
- Refinement and extension of definitions for sensor response characteristics
- Incorporation of concepts and schema developed under the OGC coordinate systems and coordinate operations group

Additional support for SensorML refinement has been provide by the National Geospatial-Intelligence Agency (NGA), the Joint Interoperability Test Command, and the OGC OWS 2.1 Project. A major part of efforts from March 2004 to October 2005 has been on blending SensorML with several ongoing standardization efforts, including particularly Transducer Markup Language (TML) (Steve Havens), ISO 19130, and the U.S. Measurements and Signals Intelligence (MASINT) standardization program. These efforts have resulted in significant refinement of SensorML, as well as improved confidence in the abilities of SensorML to meet the desired objectives. In November 2004, the SensorML document, OGC 04-019r5 was approved as an OGC Recommended Paper *[OGC 04-019r2]*.

In March 2005, OGC began the OWS 3 Interoperability Project, that allowed SensorML and the rest of the SWE components to mature and to be tested through implementation and interoperability experiments. This document is one of the products of the OWS 3 effort ending in October 2005. Before and during OWS 3, the final design of SensorML progressed from consisting of components that included measurements and processes, to modeling sensor systems primarily as process. Thus, in the current specification, SensorML defines a collection or chain processes which may include detectors, actuators, and filters, as well as other data processing elements. In November 2005, the SensorML core specification document was approved as an OGC Best Practices Document *[OGC 05-086]*. SensorML was submitted to the Request for Comment stage in May 2006 and approved as a Version 0.0 Technical Specification in July 2006.

Between July 2006 and March 2007, the SensorML Revision Working Group responded to comments received during the RFC and revised the schema accordingly. The Version 1.0 schema and documentation were released to the OGC membership for approval on May 23, 2007, and approved as Version 1.0 on June 23, 2007.

References

[FOW1998] Fowler, M. *Analysis Patterns: reusable object models*. Addison Wesley Longman, Menlo Park, CA. 1998.

[ISO19105] ISO 19115: Geographic Information – Conformance and Testing – ISO 19115

[ISO19115] ISO TC 211Geographic Information – Metadata – Implementation Specification ISO 19115 http://www.isotc211.org/pow.htm

[OGC99] The OpenGIS Abstract Specification. Topic 7: The Earth Imagery Case. OGC Document 99-107r4. http://www.opengis.org/public/abstract/99-107r4.pdf

[PAT1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: *Elements of* Reusable Object-Oriented Software. 395pp. Addison Wesley, 1995.

[UCUM] Gunther Shadow and Clement J. McDonald, 2002. The Unified Code for Units of Measure. http://aurora.regenstrief.org/UCUM/

Additional SensorML information available at http://vast.uah.edu/SensorML